NISTIR 7049

CONTAM 2.1 Supplemental User Guide and Program Documentation

George N. Walton

W. Stuart Dols

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7049

CONTAM 2.1 Supplemental User Guide and Program Documentation

George N. Walton

W. Stuart Dols

Building Environment Division
Building and Fire Research Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8633

Prepared for:

Naval Surface Warfare Center

Dahlgren, VA

October 23, 2003

U.S. Department of Commerce

Donald L. Evans, *Secretary*

Technology Administration

Phillip J. Bond, Undersecretary of Commerce for Technology

National Institute of Standards and Technology

Arden L. Bement, Jr., *Director*

ABSTRACT

CONTAM is a general purpose, multi-zone (nodal) airflow and contaminant transport analysis tool that can be used to determine inter-zone pressure differences, airflow rates and contaminant transport in complex building structures. This tool was developed by the Building and Fire Research Laboratory of the National Institute of Standards and Technology (NIST) for the analysis of building ventilation systems and has evolved and adapted to accommodate a wide range of building engineering disciplines from indoor air quality analysis to smoke management system design. This report serves two purposes: as a supplement to the CONTAMW 2.0 User Manual with explanations of the most recent enhancements to the program and to document the program. The documentation addresses both the graphical user interface (referred to herein as ContamW) and the numerical solver (referred to herein as ContamX) of version 2.1 of the program, collectively referred to as CONTAM.

Key Words: airflow analysis; atmospheric contaminant transport; building technology; computer program; contaminant analysis; design tool; indoor air quality; multizone analysis

SOFTWARE DISCLAIMER

This software was developed at the National Institute of Standards and Technology by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain. CONTAM is an experimental system. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. We would appreciate acknowledgement if the software is used.

This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified.

Certain trade names or company products are mentioned in the text to specify adequately the experimental procedure and equipment used. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment is the best available for the purpose.

NAVIGATING THIS DOCUMENT

This document is made up of three separate parts: Part 1 is relevant to the typical users of the program and Parts 2 and 3 are provided for those interested in understanding the inner-workings of the solver and graphical user interface (GUI) respectively. Part 2 also contains information related to input and output file formats that may prove useful to the more advanced users of the program. If you are viewing this document within a pdf-viewer then you can use the *Bookmarks* to navigate to various sections of the document. You can also use hyperlinks that appear throughout the document that have been defined to provide easy access to detailed information, e.g. function definitions in the ContamX program structure. After executing a link you may return to the spot where the link was called by clicking on the left arrow in the toolbar above the document, e.g., “Go to previous view” button in the Adobe Acrobat Reader as illustrated below.

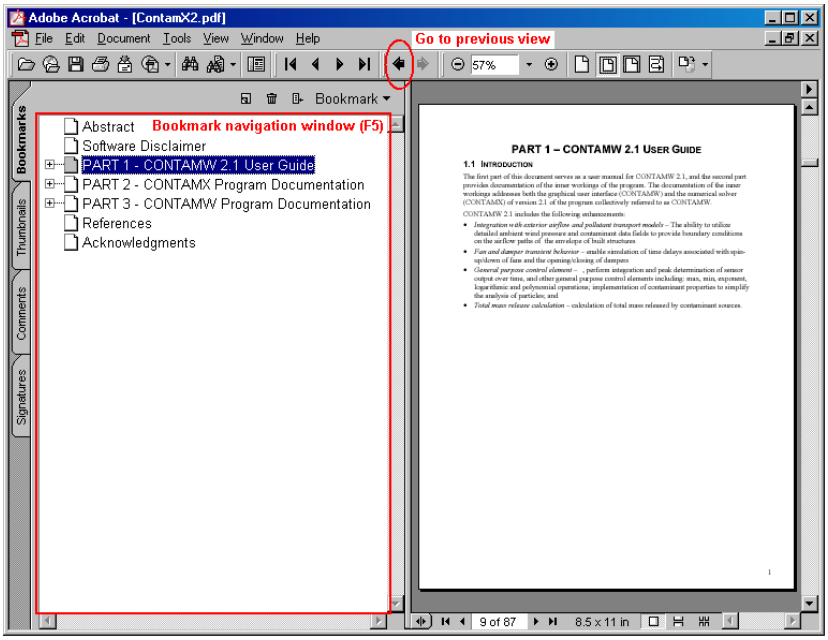


TABLE OF CONTENTS

PART 1 – CONTAM 2.1 SUPPLEMENTAL USER MANUAL

1.1 INTRODUCTION

This part of the document serves as a user manual for the enhancements embodied in CONTAM 2.1. It is to be used as a supplement to the CONTAMW 2.0 User Manual [2]. This supplemental information is also incorporated into ContamW's online help system

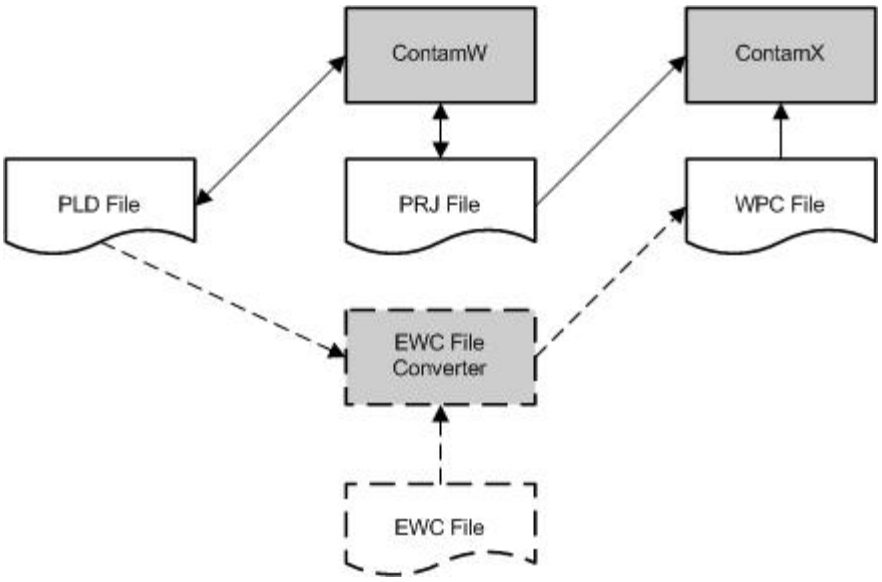
CONTAM 2.1 includes the implementation of the following enhancements:

- *Wind pressure and contaminant fields* – The ability to incorporate data from exterior airflow and pollutant transport models, e.g., plume and puff dispersion models, to utilize detailed ambient wind pressure and contaminant data fields to provide boundary conditions on the airflow paths of the envelope of built structures
- *Control elements* – New control elements to simulate time delays associated with spin-up/down of fans and the opening/closing of dampers and to perform integration, peak determination of sensor output over time, maximum, minimum and exponential operations
- *Particle analysis* – Modified contaminant properties to simplify the analysis of airborne particles
- *Mass release calculation* – The calculation of total mass released by contaminant sources during a simulation

1.2 WORKING WITH WPC FILES

A new method to account for the variation of external wind pressures and outdoor contaminant concentrations over the building envelope has been implemented in CONTAM. This method addresses the need to allow for the use of general, spatially varying wind pressure and ambient contaminant concentrations such as those from wind tunnel experiments or atmospheric models, e.g., plume or puff dispersion simulation tools. This method involves the implementation of a Wind Pressure and Contaminants file (WPC file). This file provides exterior pressure and/or contaminant concentrations time histories for every flow path that connects to the ambient zone including duct terminals and outdoor air intakes of CONTAM's simple air handling systems.

The WPC files are created externally to CONTAM, however their creation can be assisted by ContamW that creates a Path Location Data (PLD) file listing all the airflow path locations that are connected to the ambient zone. The following figure illustrates the interaction between CONTAM and the WPC file. The dashed lines in the figure represent optional components. The WPC file is an ASCII file that could be created using conventional means, e.g., spreadsheet converted to text. Once could also develop a WPC File Converter program to create the files from External Wind Pressure and Contaminant data files created by a separate tool, e.g., exterior CFD program. A converter could then work with the PLD file to create a WPC file specific to the building in question, i.e., the PRJ file. ContamW provides a means to activate a user-selectable converter. The details of the WPC and PLD file formats are presented in the sections titled Wind Pressure and Contaminant File (.WPC), and Path Location Data File (.PLD) respectively of Part 2 of this document. CONTAM does not include either an EWC file creation tool or an EWC-to-WPC converter tool.



WPC File Implementation Schematic

Steps to implement WPC files:

- Specify WPC file usage parameters (Weather→Use WPC File…)
  - Select type of data the *WPC file* includes – wind pressures and/or contaminants
  - Select existing *WPC file*
    Or
  - Select *EWC File Converter* program (optional and user-provided)
    Specify converter parameters: equivalent origin, location tolerance, date and time info
    Select *EWC* input file to *EWC File Converter* program
    Specify name of WPC file to create
- Enter C*oordinate Information* for each external flow path
  - Airflow paths
  - Duct terminals
  - Outdoor air intakes of simple air handling systems
- Run Simulation or Create ContamX Input File for batch processing. Either option will activate the WPC file implementation routines (as needed) that will create a PLD file, call the converter program, compare WPC and PLD files and notify user of discrepancies between them.

1.2.1 WPC Usage Parameters

The following parameters are required to specify the usage of a WPC file when performing a simulation. Access these parameters via the **Weather→WPC File…** menu item. The values in the group labeled *WPC File* are required, while the others are only necessary if implementing an *EWC File Converter* program.

❑ WPC File

Use this group of data to specify the .WPC file to use when performing a simulation and the type of data contained within the file. *NOTE: Check at least one of the two check boxes in order to*

activate the Coordinate Information parameters on the Airflow Path, Duct Terminal, and Simple Air Handling System property sheets.

Wind Pressures: Check this box to use wind pressures from a WPC File.

Contaminant Concentrations: Check this box to use contaminant concentrations from a WPC File. This box will only be activated if you have already defined contaminants within the current project.

Name: This is the name of the *WPC File* to use during the simulation. You can either use the *Browse…* button to select an existing file, or enter the name you would like the *EWC File Converter* to give to a new file.

Description: This will display the description line of the *WPC File* if it exists, or you can enter a description for a new *WPC File*.

- **External Wind and Contaminant Data**

  The ***E**xternal **W**ind Pressure and **C**ontaminant* file contains the external contaminant concentrations and pressures that the *EWC File Converter(which must be developed for your specific application)* will convert into a *WPC File*.

  File Name: Use the *Browse…* button to select an existing *EWC File*.

  Program to Create WPC File: Use the *Browse…* button to select the *EWC File Converter* program to use.

- **Coordinate Transformation Data**

  These parameters can be used by an EWC File Converter to establish the relationship between the coordinates of the ***E**xternal **W**ind Pressure and **C**ontaminant* file and the CONTAM project file. No transformation is required if the coordinate systems for the PRJ and EWC files are consistent. These values are stored in the PLD file by ContamW.

  Origin (X, Y and Z): The location of the origin of the CONTAM PRJ file with respect to the origin of the ***E**xternal **W**ind Pressure and **C**ontaminant* file.

  Rotation Data: The rotation of the x and y axes of the EWC coordinate system about the z axis to align with the x and y axes of the CONTAM coordinate system. Counter-clockwise is considered the positive direction.

- **Conversion Tolerance**

  These are the tolerances that an *EWC File Converter* might use to determine how closely information in the *EWC File* and the *PLD File* must match in order to resolve contaminant (species) and location data. These values are stored in the PLD file by ContamW.

  Species: Species can be resolved by their molecular weight, i.e., the molecular weight of each species defined in ContamW and the WPC file must not differ by more than this amount to be considered the same species.

  Location: Locations can be resolved by their distance as determined by an *EWC File Converter*, e.g., distance to center of a grid cell. Units within the *PLD File* are in meters.

- **Date and Time**

  These parameters can be used by an *EWC File Converter* to determine the date and time values written to the *WPC File* for the convenience of working according to CONTAM's schedules. These values are stored in the PLD file by ContamW.

  Data Time Shift: The format for this value is hh:mm:ss. An *EWC File Converter* could add this value to the *EWC File* time. For example an *EWC File Converter* could create a file that begins at 00:00:00 and set the next time to be the *Data Time Shift* you enter here. The data for these two times would be the initial values in the *EWC File*.

  Start and End Date: The format for this value is mmmdd. For example, enter January 1 as Jan01. An *EWC File Converter* could use this to output the date(s) to use when running transient simulations.

1.2.2 Envelope Opening Locations

When using *WPC Files* you must enter the coordinates for each opening connected to the ambient. CONTAMW now provides a means to do this for *Airflow Paths*, *Duct Terminals*, and *Outdoor Air Intakes of Simple Air Handling Systems*. This information is provided on the *Wind Pressure Property Pages* of the *Airflow Path Properties* and *Duct Junction Properties* and on the *AHS Property Page* of the *Simple Air Handling System Properties* input dialog boxes. The location information will be used to create the *PLD File* which in turn is used to verify there is matching information within the *WPC File* prior to performing simulations using the *WPC File*.

Even though every opening requires location data when simulating with *WPC Files*, they do not necessarily have to be unique. They simply must match a location within the WPC File.

NOTE: Either the *Wind Pressures* or *Contaminant Concentrations* check boxes on the *Wind Pressure and Contaminants (WPC) File Parameters* dialog box must be checked in order to access the coordinate input fields of these property pages.

1.2.2.1 Airflow Paths and Duct Terminals

Enter the X and Y coordinates and units for the selected airflow path or duct terminal. The Z coordinate will be calculated by ContamW to be the *Relative Elevation* of the airflow path or duct terminal plus the *Elevation* of the building level on which the path or terminal is located.

1.2.2.2 Outdoor Air Intakes of Simple Air Handling Systems

Enter the X, Y and Z coordinates and units for the outdoor air intake of the selected *Simple AHS*. Enter the Z coordinate as the height of the midpoint of the outdoor air intake with respect to the building reference height.

1.2.3 Running Simulations using a WPC File

To run a simulation using a WPC file, you must have checked either the *Wind Pressures* or *Contaminant Concentrations* check box on the *Wind Pressure and Contaminant (WPC) File Parameters* dialog box. When you choose Run Simulation (or Create ContamX Input File) from the Simulation menu, ContamW will perform a series of checks to make sure that all the path locations have been defined and match those in the WPC file if it exists and that the species

match between the PRJ and the WPC files. It will call the *EWC File Converter* and create a WPC file if an EWC file converter has been identified.

ContamW will provide error messages as needed and highlight paths on the SketchPad to reveal those for which location data are not defined or the PLD and WPC coordinates do not match within the specified tolerance. If there are errors with specific openings, you can take this opportunity to correct them.

1.3 CONTROL ELEMENTS

Several building control elements have been added to CONTAM. These elements are described below.

- **Continuous Value File (CVF)**

  This control element allows you to implement general schedules by obtaining input values from a file. This allows you to create schedules that are not restricted by the 12-*day schedule* limit of CONTAM's *week schedules*. The file is referred to as a *continuous value file*. This file is an ASCII file that you create according to the format specified in the Continuous Value File (.CVF) section in Part 2 of this document. You can only use one CVF file per simulation, and the file may contain multiple lists of values. Value lists are referenced by *Value name* which are column headings in the file.

  Value Name: Select the name of a set of values from the list of headings as they appear in the CVF file.

  File Name: This field simply displays the CVF file that contains the data that will be used by the control node during simulation. You must use the **Data→Continuous Values File…** menu item to select the file you want to use prior to creating CVF control nodes.

  Node Name: This is an optional name you can provide for this control node. This name can be used to reference this node with a *Phantom control node*.

- **Discrete Value File (DVF)**

  This control element allows you to implement scheduled discrete events by obtaining event times and values from a file. This allows you to create schedules that do not repeat according to CONTAM's *week schedules*. The file is referred to as a *discrete value file*. This file is an ASCII file that you create according to the format specified in the Discrete Value File (.DVF) section. You can only use one DVF file per simulation, and the file may contain multiple lists of events. Event lists are referenced by *Value name* which are column headings in the file.

  Value Name: Select the name of a value from the list of headings as they appear in the .DVF file.

  File Name: This field simply displays the DVF file that contains the data you need. You must use the **Data→Discrete Values File…** menu item to select the file you want to use prior to creating a DVF control node.

  Node Name: This is an optional name you can provide for this control node. This name can be used to reference this node with a *Phantom control node*.

- **Scheduled Delay**

  This control element provides the ability to simulate time delays associated with the ramping up/down of system components changing between states, e.g., the spin down of a fan or the opening/closing of a damper. The *scheduled delay* element allows you to define a schedule according to which the change of state occurs. The *output* will change according to this schedule when the *input* changes.

  Node Name: This is an optional name you can provide for this control node. This name can be used to reference this node with a *Phantom control node*.

  Schedule - Signal Increasing: Select/enter a day schedule that characterizes the delay in an increasing signal. The schedule must be *trapezoidal* beginning with a value of 0.0 at time 00:00:00 and increase to a value of 1.0 before 24:00:00. Only one increasing time period per schedule will be allowed.

  Schedule - Signal Decreasing: Select/enter a day schedule that characterizes the delay in a decreasing signal. The schedule must be *trapezoidal* beginning with a value of 1.0 at time 00:00:00 and decrease to a value of 0.0 before 24:00:00. Only one decreasing time period per schedule will be allowed.

  Description: Enter an optional description for this control element.

- **Exponential Delay**

  This control element provides the ability to simulate time delays associated with the ramping up/down of system components changing between states, e.g., the spin down of a fan or the opening/closing of a damper. The *exponential delay* element allows you to define an exponential delay based on a time constant.

  Node Name: This is an optional name you can provide for this control node. This name can be used to reference this node with a *Phantom control node*.

  Time Constants
  Increase: Enter the amount of time it should take for the *output* signal to exponentially increase by (1 – 1/e)% of the total change in the *input* signal when it rises from one state to the next. The format is hh:mm:ss.

  Decrease: Enter the amount of time it should take for the *output* signal to exponentially decrease by (1 – 1/e)% of the total change in the *input* signal when it falls from one state to the next. The format is hh:mm:ss.

  Description: Enter an optional description for this control element.

- **Maximum and Minimum**

  The *output* will be the *maximum* or *minimum* of all *input* signals to the control node each time step. Once a node is defined to be of this type, up to three input signals can be drawn directly into it. More signals can be cascaded through the use of *phantom* control elements.

  Node Name: This is an optional name you can provide for this control node. This name can be used to reference this node with a *Phantom control node*.

  Description: Enter an optional description for this control element.

- **Integrate over time**

  The *output* will be the integration over time of *input signal 1* (horizontal) controlled by *input signal 2* (vertical). Integration occurs when *input signal 2* > 0; there is no integration when *input signal 2* = 0; the integral is reinitialized to zero when *input signal 2* < 0. A simple trapezoidal integration method is used.

- **Running average**

  The *output* will be the average of the *input* signal integrated over the time span, Δt_{int}, set by the user: $out = \int_{\Delta t_{int}} in\, dt \Big/ \Delta t_{int}$. At the start of simulation, when the simulated time is less than the time span, the output will be the average of the input up to that time.

  Node Name: This is an optional name you can provide for this control node. This name can be used to reference this node with a *Phantom control node*.

  Time Span: Enter the amount of time included in the running average, Δt_{int}. The format is hh:mm:ss.

  Description: Enter an optional description for this control element.

1.4 PARTICULATE CONTAMINANTS

The properties of species/contaminants have been enhanced to simplify the simulation of particulate contaminants with CONTAM. Two properties have been added: *mean diameter* and *effective density*. Also, the units for concentration now include particle-related units. These units include *# per unit mass of air* and *# per unit volume of air*. The *mean diameter* and *effective density* are currently only utilized to convert contaminant concentrations between particle count units and mass and volumetric units.

Mean Diameter: Enter the mean particle diameter for particulate species.

NOTE: This is not necessarily meant to be the aerodynamic diameter, and that the current contaminant source models do not treat particles different from gaseous contaminants.

Effective Density: Enter a density that you want CONTAM to use as the effective density of a species you want to consider to be a particulate species.

1.5 CALCULATION OF TOTAL MASS RELEASED FROM SOURCES

The total amount of mass generated/removed by each source within a project is now calculated and output to the CONTAMX2.CSM file located in the ContamW program directory. The following shows a sample of the data output. The first two columns are the source/sink number and the zone number for referencing back to ContamW.

```
Contaminant source/sink summations.
s/s # zn #  generated  removed [kg] species
    1    1  6.000e-004  0.000e+000  C1
    2    1  6.000e-004  0.000e+000  C1
    3    1  1.000e+000  0.000e+000  C2
```

PART 2 – CONTAMX PROGRAM DOCUMENTATION

2.1 INTRODUCTION

This documentation is intended as an aid to understanding the ContamX 2.1 computer code and is meant primarily to address the logical structure of the program. The numerical algorithms are explained in the CONTAMW 2.0 User Manual [2].

The overall logical structure of the program is presented below in Section 2.3 Program Structure Overview. This material is not necessary for the general user of the program, but is provided for those interested in understanding or perhaps enhancing its capabilities. You can use the hyperlinks to jump to detailed descriptions of the highlighted functions. After executing a link you may return to the spot where the link was called by clicking on the left arrow in the toolbar above the document. This process is designed to facilitate both the rapid review of program structure and quick access to the details of the calculations. If working from a hard copy simply keep in mind that functions are presented alphabetically. There are also many small functions performing simple operations that are called from many places in the program – see Utility Functions. A special method is used to store the data for the solution of the simultaneous equations – see Sparse Matrix Data Structure. That is followed by a brief description of some of the numerical methods – see Solution of the Species Differential Equations.

2.2 DEVELOPMENT ENVIRONMENT

CONTAMX Version 2.1 was developed using Microsoft Visual C++ versions 6.0 and .NET. It is basically a console application written in the C programming language. However, the program does require Microsoft Windows in order to display simulation progress in a dialog box. Other than that, it could be compiled in a non-Windows environment with minor modifications to the source code.

2.3 CONTAMX PROGRAM STRUCTURE

This section presents the program structure of the solver, ContamX. Section 2.3.1 presents the overall program structure, 2.3.2 presents the main solver functions in alphabetical order, 2.3.3 presents the utility functions used by the solver, 2.3.4 presents the sparse matrix data structure and 2.3.5 presents the solution to the species differential equations.

2.3.1 Overall Program Structure

Program initialization [mostly from main( ) in contamx.c]
- Determine path to CONTAMX2.EXE
- Open CONTAMX2.LOG file for informative dumps
- Determine path to project file – command line or interactive input
- Determine if dialog box used
  - if so, open it and start ContamX thread,
  - otherwise, continue ContamX in DOS window
- Set output mode (dialog box or DOS window) for error messages [from contamx()]

Data initialization [from contamx( ) in contamx.c]
- Read project file – see prj_read( )
- Complete ContamX data structures – see sim_data( )

Create simulation arrays [from contamx( ); see mat_init()
- Set up airflow matrices – see af_mat_set( )
- Set up non-trace mass matrices – see mf_mat_set( )
- Set up trace mass matrices – see mf_mat_set( )
- Report equation numbers to LOG file

Initialize simulation [from contamx( ) in contamx.c]
- Set start/end dates and times
- Open/initialize weather file – see weather_init( )
- Open/initialize contaminant file – see ambtctm_init( )
- Initialize schedules and control values – see ctrl_sim( )
- Open simulation output files

Perform simulation [from contamx( ) in contamx.c]
- Initialize flow and mass fraction values
  - from restart file - resget
  - or by steady-state calculation – see sim_init( )
- Perform transient or cyclic calculation – see sim_loop( )

Normal program termination [from contamx( ) in contamx.c]
- Report simulation performance to LOG file
- De-allocate all earlier heap allocations – see mat_term( )
- Perform memory checks
- Close LOG file
- Return to main(); return to operating system

2.3.2 ContamX Solver Functions

- **af_mat_alc( )**
  [function in file matset.c]

Allocate remaining airflow solution arrays.

Determine number of non-zero coefficients in symmetric matrix – in ija_minset( )
Allocate and set the *ija* index vector – in ija_minset( )
Allocate the *sa*, *b*, and *x* coefficients vectors
For symmetric skyline solution method allocate remaining arrays – in sky_alc( )
For conjugate gradient method allocate *p, pp, r, rr, z, zz,* and preconditioning vectors
For Gauss elimination allocate *aa* full matrix

- **af_mat_set( )**
  [function in file matset.c]

Set up airflow matrices.

Transfer control parameters from `_rcdat` to the structure for airflow calculations
Set up vector of pointers to nodes with constant pressure nodes last
Determine positions of all non-zero elements in flow solution matrix – in acolset( )
Allocate and fill the *ijf* vector sparse data mapping – see Sparse Matrix Data Structure
For the skyline method for solving simultaneous algebraic equations
Report the initial matrix profile structure – in ija_prfl( )
    Compute new sequence of equations to reduce average profile – in ija_optord( )
    Reset the airflow equation numbers in the node data structures
    Report the final matrix profile structure
Allocate remaining arrays depending on solution method – see af_mat_alc( )
Set pointers for off-diagonal elements in the link structures
Report solution notes to the LOG file

- **ambtctm_get( )**
  [function in file weather.c]

Get the weather file data for the current simulation time.

Note: the _ambtctm structure defined in weather.c holds data for computing the contaminant mass fractions at the current simulation time (_sim_time) from data in the contaminant file at the times (time0 and time1) that bound the _sim_time.

Set the contaminant file pointer
Reset time at end-of-day
Read the contaminant file until `time0` < `_sim_time` <= `time1`
    Reset `time0` and corresponding data as necessary
    Check the data date
    Read and store the time and corresponding contaminant data
Determine the mass fraction of the first non-trace contaminant
Set contaminant data for current simulation; use linear interpolation if not at `time1`
Transfer data to the structure for `_ambt` node global variable

- **ambtctm_init( )**
  [function in file weather.c]

Initial processing of the contaminant file.

Open the contaminant file – using nxtopen( )
Confirm file type and version
Read the contaminant file start and end dates
Confirm simulation dates within contaminant file dates
Read number of species in file and allocate vectors in _ambtctm structure
Read species names and determine which match simulation contaminants
Position contaminant file at the simulation start date
Read and store contaminant data for time 00:00:00
For cyclic simulation save position in contaminant file of first time step
Determine the mass fraction of the first non-trace contaminant

The weather and contaminant files are read using the nxtword( ) utility function. This requires that the _unxt file pointer be set to the weather file or the contaminant file before each is read.

- **calc_SP( )**
  [function in file simulate.c]

Calculate stack pressures for each flow path.

This function computes the P_1, P_2, and $\rho g(z_1-z_2)$ terms from the Bernoulli equation:

$$\Delta P = \left(P_1 + \frac{\rho V_1^2}{2} \right) - \left(P_2 + \frac{\rho V_2^2}{2} \right) + \rho g (z_1 - z_2)$$

P_1 and P_2 are computed by a hydrostatic adjustment to the reference pressures in nodes 1 and 2. z_1 and z_2 are the elevations of the ends of the link relative to nodes 1 and 2. For airflow paths the elevations are identical. They may differ for ducts. The algorithm includes hydrostatic equations for incompressible and compressible air.

The final result is an estimate for the pressure drop across each opening based on the reference pressures in the connected nodes, the wind pressure, and the 'stack' pressure.

- **calc_WP( )**
  [function in file simulate.c]

Calculate wind pressures for each flow path.

The wind pressure for links not connecting to the ambient node is zero. It is also zero if no wind pressure model has been specified for the link. Wind pressure models include constant value, linear interpolation using lint1d( ), spline interpolation using splint( ), and a trigonometric interpolation using htrigf( ), between user specified wind pressure values as a function of relative azimuth angle.

- **ctrl_links( )**
  [function in file prjdata.c]

Set control nodes and links for simulation.

Set pointers to incoming data according to data type and number
Create linked list of control nodes in calculation sequence

Here it is necessary to set the pointers in sensors to the proper node data (for zones and junctions) and the proper link data (for paths and ducts). Note that nodes and links can be sensed and controlled making it is impossible to place the control data in the project file so that the referenced items have been defined.

The control structures are then placed in a linked list (starting at _Ctrl0) so that when a node is called while sequentially processing the list in ctrl_sim( ) the input signals will be current. The sequence was determined in ContamW in function ctrl_order( ) in file prjpack.c.

- **ctrl_sim( )**
  [function in ctrlsim.c]

Process day-schedules, exposure-schedules, and controls.

Set clock-time (incl. DST) and day type
Loop through the week-schedule list
Loop through the control nodes list
Set any scheduled zone temperatures
Loop through the exposure-schedule list

- **de_mat_alc( )**
  [function in file matset.c]

Allocate differential equation solution arrays.
Determine number of non-zero coefficients in symmetric matrix – see ija_minset( )
Allocate and set the *ija* index vector – see ija_minset( )
Allocate the *sa*, *b*, and *x* coefficients vectors
For symmetric skyline solution method allocate remaining arrays – in sky_alc( ) in file matset.c
For conjugate gradient method allocate *p, pp, r, rr, z, zz,* and preconditioning vectors
For Gauss elimination allocate *aa* full matrix.

- **FillAf( )**
  [function in afesim.c]

Fill the Jacobian matrix for air flows.

Clear the Jacobian coefficients vector
Loop through all air links
Initialize the coefficients on the diagonal to dM/dt for variable pressure nodes
Initialize the coefficients on the diagonal to 1.0 for variable pressure nodes
Initialize the ∑ F and ∑ |F| values for each node.
Loop through all air links
    If a numerical derivative is being computed
        Compute flows for adjusted pressure drop
    For each flow (1 or 2) through the link
        Compute dF/dP if a numerical derivative is used
        Add flows to the ∑ F and ∑ |F| values

Add dF/dP value to the appropriate coefficients in the Jacobian vector
Check for zeroes on the diagonal (would cause divide-by-zero in solution)
Set any such nodes to constant pressure and recompute all values

Note the use a linked list of link (path) structures and of C pointer to functions:

```
for( pp=_pafp0; pp; pp=pp->next)  // loop through all flow paths
  {
  IX (*pf)(AF_PATH ) = pp->pe->pfunc; // pointer to simulation function
  . . .
  pf( pp );  // compute the flow(s) through path pp
  . . .
```

The linked list provides a very simple structure to loop through the flow paths. The pointers to functions provide a way to call the different types of flow paths without creating a switch statement to access each different type of path. The functions to evaluate each different type of flow path or duct are included in file afesim.c.

- **Fill_Mf( )**
  [function in solvmf.c]

- **ija_minset( )**
  [function in file matset.c]

Minimize the sparse matrix index vector.

Reduce the "full" sparse matrix index vector, *ijf*, to *ija* which does not include diagonal elements or upper triangle, if symmetric. Call once with count = 0 to determine size of *ija*, then call with count = 1 to fill *ija*.

- **ija_optord( )**
  [function in file matset.c]

Determine the optimum (minimum profile) ordering for a set of simultaneous equations.

From the old sparse matrix index vector *ija* (see Sparse Matrix Data Structure) use the ACM Transactions On Mathematical Software (TOMS) 582 algorithm to compute a new optimum ordering for the variable pressure/mass/… rows. The TOMS 582 algorithm has been converted from Fortran to C in file gpskc.c. Return the new ordering in the *new* vector. Return 1 if the ordering has changed, 0 if it has not.

- **init_Af( )**
  [function in solvaf.c]

Use linear relations to determine initial guess for pressures.
Zero the sparse matrix array
Loop through all links setting airflow matrix coefficients for linear flow elements models
Solve the simultaneous linear equations for node pressures – see solve_slae( )

Set up the coefficients of the airflow matrix so that solution of the simultaneous linear equations will give an initial guess of pressures to start the Newton-Raphson method for solving the non-

linear airflow equations. This initialization tends to save a few N-R iterations and is useful for cases involving large numbers of airflow nodes.

- **init_Mfn( )**
  [function in solvmf.c]

Initialize arrays for non- trace mass fraction calculations.
Set the initial mass fractions in the zones and junctions.
Set the initial mass fractions in the boundary layer controlled sinks.
Set up the matrix for computing the mass fractions – in Fill_Mf( )

- **init_Mft( )**
  [function in solvmf.c]

Initialize arrays for trace mass fraction calculations.
Set the initial mass fractions in the zones and junctions.
Set the initial mass fractions in the boundary layer controlled sinks.
Set up the matrix for computing the mass fractions – in Fill_Mf( )

- **mat_init( )**
  [function in file contamx.c]

Allocate the matrices for solving airflows and mass fractions.
Allocate _facBins to report airflow solution iterations
Set up airflow matrices – see af_mat_set( )
Set numerical derivative flags – in setNmDrv( )
If using non-trace contaminants
    Copy control parameters from _rcdat to _MFn_mat
Set up non-trace mass matrices – see mf_mat_set( )
If using trace contaminants
    Copy control parameters from _rcdat to _MFt_mat
Set up trace mass matrices – see mf_mat_set( )

- **mat_term( )**
  [function in file contamx.c]

Free the allocated matrices for solving airflows and mass fractions.

All heap memory allocated in mat_init( ) is freed in reverse order in mat_term( ). Other memory allocated earlier in prj_read( ), contamx( ), and main( ) is also freed.

- **mf_mat_set( )**
  [function in file matset.c]

Set up contaminant matrices.
Transfer control parameters from _rcdat to the structure for mass fraction calculations
Set up vector of pointers to nodes with constant pressure nodes last
Determine positions of all non-zero elements in flow solution matrix – in acolset( )
Allocate and fill the *ijf* vector sparse data mapping – see Sparse Matrix Data Structure

For the skyline method for solving simultaneous algebraic equations
    Report the initial matrix profile structure – in ija_prfl( )
    Compute new sequence of equations to reduce average profile – see ija_optord( )
    Reset the airflow equation numbers in the node data structures
    Report the final matrix profile structure – in ija_prfl( )
Allocate remaining arrays depending on solution method – see de_mat_alc( )
Set pointers for off-diagonal elements in the link structures
Report solution notes to the LOG file

- **prj_read( )**
  [function in file prjread.c]

Read the project (PRJ) file.

This and related functions in the same file have been written to read the project file for ContamW or for ContamX depending on the definition of the macro CTMW being 1 or zero, respectively. ContamW uses data such as sketchpad data and units for displaying parameters that are not needed in ContamX.

The ContamW version of prj_read( ) includes code for updating previous versions of the project file. Sections of the project file are read by functions within the prjread.c file. Sections are ordered so that items referenced by pointers in the data structures are read before they are referenced. For example, a flow path must refer to a flow element, so flow elements are read before flow paths. If a significant error is encountered while reading any section, input processing is terminated and an error code is returned by prj_read( ).

The ContamW version calls functions to check and process pointers in the controls because control nodes have pointers to other control nodes. The ContamX version includes related processing for controls and functions to create the airflow network by converting zones and junctions to network nodes and converting paths and ducts to network links – see sim_data( ).

Several temporary data structures are allocated, used, and then freed. For example, most named elements such as schedules, filters, flow elements, etc., are stored in individually allocated structures with an allocated vector of pointers to those structures. When such an element is later referenced by sequence number, that number is converted to the pointer to the element according to the vector of pointers.

- **sim_data( )**
  [function in file prjdata.c]

Convert project data to forms used for simulation.
Transfer elements from the _rcdat structure to individual global variables
Transfer steady-state weather data to ambient airflow node.
Transfer default contaminant concentrations to ambient airflow node.
Set ductwork junction volumes – in jct_set( )
Set airflow node data – in afnd_set( )
Set airflow link data – in afpt_set( )
Define implicit flow elements: simple AHS and duct leaks
Set implicit flow paths

Check completeness of airflow network – in airnet_check( )
Create linked list of day-schedules
Create linked list of week-schedules
Complete trance and non-trace source linked lists
Copy boundary-layer non-trace sinks to separate list
Copy boundary-layer trace sinks to separate list
Convert occupancy zones to node pointers
Create linked list of exposure structures
Complete control links – see ctrl_links( )

- **sim_init( )**
  [function in file simulate.c]

Initialize flows and compute steady-state mass fractions.

If using non-trace contaminants
    Compute gas constant for mixed air in each zone
    Compute mass of air in each zone
If using WPC File
    Get wind pressure from WPC File [in WPC_get()]
Compute wind pressures – see calc_WP( )
Compute stack pressures – see calc_SP( )
Initial guess for pressures by linear flow approximation - see init_Af( )
Reset convergence and time step values
Iterate through density changes:
    {
    Solve for airflows by Newton-Raphson iteration with
    simple trust region method – see SolveAfstr( )
    or simple under-relaxation – see SolveAfsur( )
    Break loop if densities have converged
    Tighten N-R convergence for next iteration
    If using non-trace contaminants
        Compute non-trace mass fractions [in init_Mfn() and solv_DE()]
    Compute gas constant for mixed air in each zone
    Compute mass of air in each zone
    Re-compute stack pressures - see calc_SP( )
    }
Initialize trace and non-trace mass transfer data – see init_Mfn( ) and init_Mft( )
Compute trace mass fractions [call solv_DE()]
If not initializing for transient simulation, report results – in simout( )

- **sim_loop( )**
  [function in file simulate.c]

Perform transient simulation.

Set parameters to control output and time step loop
Display information (for DOS window only)

```
While (not done):
        {
        Check for/process user interruption
        Initialize daily summary results as needed
        Increment time of day by one time step
        Get contaminant data - see ambtctm_get( )
        Get weather data – see weather_get( )
        Compute airflows and mass fractions for current time – see sim_step( )
        Write results if time of day is on a listing time step – in simout( )
        Compute flows for daily summary
        Compute mass fractions for daily summary
        Check for reaching non-cyclic end point – set done
        If (time = 24:00:00)
                Write daily summary results
                Check for cyclic convergence – set done
                If  (not done)
                        Advance day count; reset time of day to 00:00:00
                        Write results
                        Reset read position in weather and contaminant files
        }
Write summary results – in simout( )
Close results files, weather file, and contaminant file
```

- **sim_step( )**
  [function in file simulate.c]

Simulate a single time-step.

```
Save mass and mass fraction values at end of last time step
Set all schedule values and process controls - see ctrl_sim( )
Compute scheduled mass gains
Compute wind pressures - see calc_WP( )
Loop until zone masses converge
        {
        Compute stack pressures - see calc_SP( )
        Solve for airflows by Newton-Raphson iteration with
                simple trust region method – see SolveAfstr( )
                or simple under-relaxation – see SolveAfsur( )
        If using non-trace contaminants:
                Compute non-trace mass fractions – see solve_Mfn( )
        Re-compute gas constant for mixed air in each zone
        Compute mass in each zone
        Check zone mass convergence
        }
Solve trace species mass fractions – see solve_Mft( )
Compute exposure values
```

- **solve_DE( )**
  [function in file solvde.c]

Solve differential equations by trapezoidal integration and mixed direct/iterative solution of simultaneous equations.

Solve [A] {x} = {b}. Temporary [A] and {b} created so new {xp} can be calculated.

- **Solve_Mfn( )**
  [function in solvmf.c]

Solve for the mass fractions of the non-trace contaminants.

This function uses the same process described in solve_Mft( ) except that only the non-trace contaminants are considered.

- **Solve_Mft( )**
  [function in solvmf.c]

Solve for the mass fractions of the trace contaminants.

See also Solution of the Species Differential Equations.

If on the first iteration during a time step:

    Compute the constant (predictor) portions of the difference equation

Fill the [A] matris – in Fill_Mf( )

Solve the difference equations for values at the end of the time step

Transfer results to node and sink structures

- **SolveAfstr( )**
  [function in solvaf.c]

Solve symmetric simultaneous non-linear algebraic equations by Newton-Raphson with simple trust region.

Dr. David Lorenzetti developed this function. It provides a more robust algorithm than the earlier under-relaxation method. It is the default method and should be used unless some particular problem is encountered.

- **SolveAfsur( )**
  [function in solvaf.c]

Solve symmetric simultaneous non-linear algebraic equations by Newton-Raphson with simple under-relaxation.

Loop until convergence or iteration limit

    Fill the airflow solution matrix – see FillAf( )
    Check convergence
    Solve simultaneous linear equations for dP values - see solve_slae( )
    Under-relax the pressure adjustments
    Adjust the airflow node pressures

Clear any numerically insignificant airflows

This method was used in ContamW 1.0 and the earlier DOS versions of CONTAM.

- **solve_slae( )**
  [function in solvse.c]

Solve symmetric simultaneous linear algebraic equations.

Select solution method:

Skyline:

- Transfer data from sparse to skyline arrays – in fill_sky_s( )
- L-U factor skyline matrix – in luf_sky_s( )
- Solve simultaneous equations directly – in lus_sky_s( )

Preconditioned conjugate gradient (PCG):

Transfer data from sparse to preconditioning array – in fill_ssm( )

- Perform incomplete Cholesky decomposition – in luf_chl_c( )
- Solve simultaneous equations iteratively – in sa_pcg( )

Gauss elimination:

- Transfer data from sparse to full arrays – in fill_ges( )
- L-U factor the full matrix – in luf_ge( )
- Solve the simultaneous equations – in lus_ge( )

The skyline method does a direct solution using a reduced array. It is more reliable than PCG and usually faster for small problems. The PCG method is faster for large problems if it converges. The gauss elimination method is only to provide a well-established solution for comparison with the other methods. It is prohibitively slow for larger problems.

- **weather_get( )**
  [function in file weather.c]

Get the weather file data for the current simulation time.

Note: the _weather structure defined in weather.c hold data for computing the weather data at the current simulation time (_sim_time) from data in the weather file at the times (time0 and time1) that bound the _sim_time.

Set the weather file pointer

Reset time at end-of-day

Read the weather file until time0 < _sim_time <= time1

- Reset time0 and corresponding data as necessary
- Check the data date
- Read and store the time and corresponding data
- Reset wind direction for interpolation

Set weather data for current simulation; use linear interpolation if not at time1

Convert ASHRAE humidity ration to Contam mass fraction of H2O

Transfer data to global variables

- **weather_init( )**
  [function in file weather.c]

Initial processing of the weather file.

Open the weather file – using nxtopen( )

Confirm file type and version

Read the weather file start and end dates
Confirm simulation dates within weather file dates
Read and store day-of-week, day-type, DST, and Tg for all dates
Position weather file at the simulation start date
Set day-data globals: _daytyp, _dayofw, _DSTind
Read and store weather data for time 00:00:00
For cyclic simulation save position in weather file of first time step

The weather and contaminant files are read using the nxtword( ) utility function. This requires that the _unxt file pointer be set to the weather file or the contaminant file before each is read.

2.3.3 Utility Functions

Compiler dependent functions have been grouped in file **config.c**. Memory allocation and de-allocation functions are in file **heap.c**. Other utility functions are in file **utils.c**. Three different compilers have been used – two for the earlier DOS-based CONTAM93 and CONTAM96 programs, and Visual C++ versions 6.0 and .NET for the current MS Windows based program.

- **File path processing**
  [functions in file config.c]

These functions are used to create output file paths that will open in the proper directory.

pathsplit( ) converts a path into its component parts (drive, directory, name, and extension).

pathmerge( ) does the opposite.

- **Error handling**
  [functions in file config.c]

Error messages are generated by a call to the **error( )** function. Passed parameters include the *severity* of the error, the name and line number of the file from which **error( )** is called, and an indefinite number of strings describing the error. The function maintains a count of 'severe' errors. The global variable _emode (in setEmode( )) controls display of the error message.

If *severity* >= 0, the function will merge the message strings into a single string and create another string reporting the file name and line number – i.e., the source of the error. The severity, source, and error message are then displayed in a dialog box or the DOS window depending on _emode, and they are written to the LOG file.

If *severity* > 2, ContamX will be terminated by calling **finish( )** which also closes all open files.

If *severity* = 2, the count of severe errors is incremented.

If *severity* < -1, the count of severe errors is reset to zero.

If *severity* = -1, the only action is to return the current count of severe errors.

If the count of severe errors reaches 10 and the dialog box is in effect, the user will be asked if he wants to terminate the simulation. If he does not, the count of severe errors is reset to zero.

This prevents an endless display of error messages that could occur in some circumstances.

The **lognote( )** function is used to write similar informative messages to the LOG file only. It is a debugging tool not intended to be informative to the general ContamX user.

- **Console input**
  [functions in file config.c]

getkey( ) is used to read a single keystroke. The *wait* parameter tells **getkey( )** to wait until a key is pressed or to return immediately if a key has not been pressed.

noyes( ) obtains a no or yes response to a query. It returns 0 for no or 1 for yes. The query is presented in a dialog box or the DOS window depending on _emode.

- **Heap tests**
  [functions in file config.c]

memrem( ) reports the memory still available in the heap. It is useful for determining if heap memory has been properly freed. It works with the Borland and Watcom compilers but is not available in Visual C++.

memwalk( ) reports the status of the allocated heap memory and then displays the status of every heap allocation. This seems to not be working with the current Visual C++.

nptest( ) reports if a value has been written to address 0. This occurs when a value is written to a null pointer. The test is not applicable to Visual C++.

- **Math error processing**
  [functions in file config.c]

flterr( ) and **matherr( )** trap various floating point errors, print error messages, and terminate the program.

- **Heap processing**
  [functions in file heap.c]

All memory allocations and de-allocations should go through alc_e( ) and fre_e( ) to allow some useful heap checking options based on the definition of the macro MEMTEST: 1 = test guard bytes; 2 = log actions; and 0 = no tests. When MEMTEST > 0, four guard bytes are added before and after the normal heap memory allocation. These guard bytes are used to test writes and reads beyond the ends of the allocated vector -- especially useful for off-by-one indexing. Based on an idea and code by Paul Anderson, "Dr. Dobb's C Sourcebook", Winter 1989/90, pp 62 - 66, 94. When MEMTEST > 1, every allocation and de-allocation is reported in the LOG file.

alc_e( ), **fre_e( )**, and **chk_e( )** allocate, de-allocate, and check the guard bytes of a single block of memory. The check is automatically performed at de-allocation.

alc_mc( ), **fre_mc( )**, and **chk_mc( )** allocate, de-allocate, and check a rectangular matrix (2-dimensional array) given minimum and maximum row and column indices. Memory is allocated contiguously.

alc_mvc( ), **fre_mvc( )**, and **chk_mvc( )** allocate, de-allocate, and check a matrix with variable length rows (used to store skyline method coefficients).

alc_v( ), **fre_v( )**, and **chk_v( )** allocate, de-allocate, and check a vector (2-dimensional array) given the minimum and maximum indices.

alc_ec( ), **fre_ec( )**, and **alc_eci( )** speed and simplify allocation and de-allocation for many small structures such as the Contam zones, paths, schedules, flow elements, etc. Small structures are 'allocated' within a larger allocated block which is initially created with alc_eci( ). A single call to fre_ec( ) will de-allocate the entire block; individual structures cannot be freed. This saves quite a bit of memory allocation overhead and is quite a bit faster than calling alloc( ) for each small structure. Based on an idea and code by Steve Weller, "The C Users Journal", April 1990, pp 103 - 107.

- **Text File Reading Functions**
  [functions in file utils.c]

nxtopen( ), **nxtline( )**, **nxtwrd( )**, and **nxtclose( )** are used to process the project, weather, and contaminant ASCII text data files. These functions process the file designated FILE *_unxt. A key feature of these functions is their handling of erroneous input values.

- **Format Conversion Functions**
  [functions in file utils.c]

dblcon( ) converts a string to an 8-byte real variable.

fltcon( ) converts a string to a 4-byte real variable.

longcon( ) converts a string to a 4-byte integer variable.

intcon( ) converts a string to a default size integer variable.

timecon( ) converts a string (hh:mm:ss) to seconds past midnight.

datecon( ) converts a string (dd/mm, mm/dd) to a day-of-year number (1 - 365).

datxcon( ) converts a string (mm/dd) to a day-of-year number (1 - 365) (for Excel compatibility).

fltstr( ) converts a 4-byte real variable to a string.

intstr( ) converts a 4-byte integer variable to a string.

timestr( ) converts a 4-byte integer (seconds past midnight) to a string of the form: hh:mm:ss.

datestr( ) converts a day-of-year integer (1 - 365) to a string (ddmm).

datxstr( ) converts a day-of-year (1 - 365) to a string (dd/mm) (for Excel compatibility).

- **Input Conversion Functions**
  [functions in file utils.c]

readR8( ) reads the next word from file _unxt and converts it to an 8-byte real variable.

readR4( ) reads the next word from file _unxt and converts it to a 4-byte real variable.

readI4( ) reads the next word from file _unxt and converts it to a 4-byte integer variable.

readI2( ) reads the next word from file _unxt and converts it to a default size integer variable.

readHMS( ) reads the next word from file _unxt and converts it to a 4-byte integer time value.

readMD( ) reads the next word (ddmm, mmdd) from file _unxt and converts it to a day-of-year number (1 - 365).

readMDx( ) reads the next word (dd/mm) from file _unxt and converts it to a day-of-year number (1 - 365).

The "default size" integer is 2-bytes for a 16 bit compiler and 4-bytes for a 32 bit compiler. CONTAM presently enforces the rule that such an integer must be in the range -32767 to +32767, i.e., fits in a 2-byte integer. This limits the maximum number of zones, paths, etc., to 32767. Relaxing this limit will also require the conversion of small integers in the data structures.

2.3.4 Sparse Matrix Data Structure

Calculation of both the airflows and mass fractions involve solving simultaneous linear algebraic equations. These simultaneous equations can be expressed in matrix form: **[A] {x}** = **{b}** where **{b}** is an N-element vector, **[A]** is an N by N matrix, and the solution **{x}** is an N-element vector. As N gets large (CONTAM projects with over 3000 nodes have been created) the size of **A** becomes excessive and the execution time using full matrix methods becomes even more excessive. When N is large only a small portion of elements of **A** are non-zero, therefore sparse matrix techniques should be used to solve such problems. There are also methods to store only the non-zero elements of **A.**

Press et al. [8 p78] present a good method for storing sparse matrices:

"To represent a matrix **A** of dimension N × N, the row-indexed scheme sets up two one-dimensional arrays, call them *sa* and *ija*. The first of these stores matrix element values in single or double precision as desired; the second stores integer values. The storage rules are:

· The first N locations of *sa* store **A**'s diagonal matrix elements in order. (Note that diagonal elements are stored even if they are zero; this is at most a slight storage inefficiency, since diagonal elements are nonzero in most realistic applications.)

· Each of the first N locations of *ija* stores the index of the array *sa* that contains the first off-diagonal element of the corresponding row of the matrix. (If there are no off-diagonal elements for that row, it is one greater than the index in *sa* of the most recently stored element of a previous row.)

· Location 1 of *ija* is always equal to N + 2. (It can be read to determine N.)

· Location N + 1 of *ija* is one greater than the index in *sa* of the last off-diagonal element of the last row. (It can be read to determine the number of nonzero elements in the matrix, or the number of elements in the arrays *sa* and *ija*.) Location N + 1 of *sa* is not used and can be set arbitrarily.

· Entries in *sa* at locations ≥ N + 2 contain **A**'s off-diagonal values, ordered by rows and, within each row, ordered by columns.

· Entries in *ija* at locations ≥ N + 2 contain the column number of the corresponding element in *sa*."

In ContamX the coefficients for the airflow Jacobian and for the mass fraction calculations are first stored into this sparse structure. They may then be transferred to another data structure

designed for the solution algorithm being employed. There is a small run-time overhead for this process, but it is very flexible when it is not known that a single best solution method exists for all cases.

2.3.5 Solution of the Species Differential Equations:

The transient evolution of the node contaminant masses and mass fractions are determined in a the following manner. It is expressed in terms of $\{Q\}$

$$\{Q\} = [C]\{X\} \tag{1}$$

where $\{Q\}$ is the vector of masses of all species in all nodes, $[C]$ is a diagonal matrix of node air masses, and $\{X\}$ is the vector of mass fractions.

The basic time discretization is

$$\{Q\}_t \approx \{Q\}_{t-\Delta t} + \{\dot{Q}\}\Delta t \tag{2}$$

where the following linear approximation will be used:

$$\{\dot{Q}\} = [K]\{X\} + \{G\} \tag{3}$$

$[K]$ is the matrix of inter-nodal mass flows (and reactions), and $\{G\}$ is the vector of net mass generation rates. For large problems $[K]$ will be sparse.

Equation (3) will be solved using a trapezoidal approximation of the derivative term:

$$\{Q\}_t \approx \{Q\}_{t-\Delta t} + (1-\gamma)\Delta t\{\dot{Q}\}_{t-\Delta t} + \gamma\,\Delta t\{\dot{Q}\}_t \tag{4}$$

where γ is a parameter controlling the stability and accuracy of the integration:

γ = 0 corresponds to the standard explicit method,

γ = 1/2 corresponds to the Crank-Nicholson method,

γ = 2/3 corresponds to the Galerkin method, and

γ = 1 corresponds to the standard implicit method.

γ must be greater than or equal to 0.5 for unconditional stability, although values below about 0.75 may show oscillations.

Equation (4) can be rearranged as

$$\{Q\}_t \approx \{\tilde{Q}\}_{t-\Delta t} + \gamma\,\Delta t\{\dot{Q}\}_t \tag{5}$$

where

$$\{\tilde{Q}\}_{t-\Delta t} = \{Q\}_{t-\Delta t} + (1-\gamma)\Delta t\{\dot{Q}\}_{t-\Delta t} \tag{6}$$

is a quantity based entirely on values known at end of the last time step, time t-Δt.

Equation (5) can be converted into the form to solve for $\{X\}$:

$$[A]\{X\} = \{B\} \tag{7}$$

with

$$[A] = [C]_t - \gamma\,\Delta t[K]_t \tag{8}$$

and

$$\{B\} = \{\widetilde{Q}\}_{t-\Delta t} + \gamma \Delta t \{G\}_t \tag{9}$$

In equation (8) there appears to be the possibility of a coefficient on the diagonal of [A] going to zero. This should not happen because $k_{n,n} < 0$ (and $k_{n,m} \geq 0$) which forces $a_{n,n} > 0$ (and $a_{n,m} \leq 0$). The only possible problem is in the case of an isolated node, $k_{n,n} = 0$, that is also massless, $c_n = 0$, which will require special handling.

Note on derivation of equations (7) - (9) from equation (5):

$$[C]_t \{X\}_t = \{\widetilde{Q}\}_{t-\Delta t} + \gamma \Delta t \left([K]_t \{X\}_t + \{G\}_t\right) \tag{10}$$

$$\left([C]_t - \gamma \Delta t [K]_t\right)\{X\}_t \approx \{\widetilde{Q}\}_{t-\Delta t} + \gamma \Delta t \{G\}_t \tag{11}$$

2.3.5.1 Special Cases

(1) A steady state solution for $\{X\}$ is obtained by rearranging equation (3):

$$[K]\{X\} = -\{G\} \tag{12}$$

(2) $c_n = 0$ (massless node)

There is no transient storage effect, therefore x_n at time t is determined by the flows into the node at that time. This is equivalent to the steady-state solution of equation (12).

(3) $k_{n,n} = 0$ (no flow - isolated node)

For $c_n > 0$, the c_n term in the coefficient on the diagonal (see equation (11)) will prevent division by zero.

For $c_n = 0$, solution of equation (11) will lead to a division by zero. The practical solution is to alter the coefficients in [A] and {B} so that $x_{n,t}$ will be $x_{n,t-\Delta t}$ after solving equation (7). That is, $a_{n,n} = 1$, $a_{n,m} = 0$ for all m, and $b_n = x_{n,t-\Delta t}$.

2.3.5.2 Numerical Methods

ContamX 2 includes several methods for solving equation (7), which represents a set of equations that are linear, non-symmetric, and diagonally dominant. The direct solvers are Gauss elimination (for test purposes only) and a non-symmetric skyline method for practical computations. The performance of the latter can depend on the ordering of the equations. It uses the TOMS 586 algorithm to improve that ordering. This is possible because the structure of $[A]$ is symmetric even though the coefficients are not. Two iterative methods are available: a bi-conjugate gradient technique which currently has a problem when {X} = 0, and a simple successive over-relaxation (SOR) method. The iterative methods require less memory than the direct methods, but may or may not be faster depending on the matrix size, sparsity pattern, and the number of iterations for convergence.

2.4 CONTAM INPUT AND OUTPUT FILES

This section provides detailed information of the files used by ContamX and ContamW.

2.4.1 Project File (.PRJ)

CONTAM 2 consists of two separate programs: ContamW provides the graphic user interface for describing the building and viewing simulation results; ContamX performs the simulation. "Project" files (.PRJ extension) provide input to both ContamW and ContamX. Project files are created by the functions in file prjsave.c in ContamW and read by the functions in file prjread.c in ContamW and ContamX. The project file contains some information that is used only by ContamW and not by ContamX, e.g., input units and the SketchPad display data. These data are read when the macro name CTMW is defined to be 1, and they are not read when it is defined to be 0.

Most of the data are read into structures that are defined in the files contam.h and celmts.h for ContamW and files simdat.h and selmts.h for ContamX. Some data are read into global variables that are defined in cglob.h and cxtrn.h for ContamW and sglob.h and sxtrn.h for ContamX. These variables are indicated by a leading underscore, _, in their name.

One important feature of the project files is the tremendous variety of data stored. It has been divided into sections grouping similar kinds of data, or objects, together. Each section is terminated with the special value –999 which serves as a check for some reading errors. Within each section there are usually multiple objects, each stored into an individual structure. Such a section begins with the number of objects and starts the data for each object with a sequence number that serves to check for reading errors. These objects may be ordered as arrays when the total number is constant, or as linked lists to allow the number to vary.

In the following material individual variables are briefly described. Units are indicated within brackets, [ ]. The type of variable is in parentheses (matching the definitions in the include file types.h).

I2 – a two-byte long signed integer (C type "short")

I4 – a four-byte long signed integer (C type "long")

IX – a default length signed integer (C type "int")

UX – a default length unsigned integer (C type "unsigned")

R4 – a four-byte long real (C type "float")

R8 – an eight-byte long real (C type "double")

Variables used only by ContamW are indicated by {W}.

Each section includes sample data from the *Sample21.prj* file. This file includes samples of all the different flow and source element types. Only part of Section 3 is shown (…) to save space. A simulation of the *Sample21.prj* project file can be run, but the results are not very meaningful.

Comments are included in the project file to make it somewhat human-readable. Anything after an exclamation mark, !, to the end of the line is not read by the input processor.

2.4.1.1 Sections of the PRJ file:

Section 1: Project, Weather, Simulation, and Output Controls

Section 2: Species and Contaminants

Section 3: Level and Icon Data

Section 4: Day Schedules

Section 5: Week Schedules

Section 6: Wind Pressure Profiles

Section 7: Kinetic Reactions

Section 8: Filters

Section 9: Source/Sink Elements

Section 10: Airflow Elements

Section 11: Duct Elements

Section 12: Control Nodes

Section 13: Simple AHS

Section 14: Zones

Section 15: Initial Zone Concentrations

Section 16: Airflow Paths

Section 17: Duct Junctions

Section 18: Initial Junction Concentrations

Section 19: Duct Segments

Section 20: Sources/Sinks

Section 21: Occupancy Schedules

Section 22: Exposures

Section 23: Annotations

❑ Section 1: Project, Weather, Simulation, and Output Controls

Comment lines are added by ContamW (see example) to help visually identify the parameters.

The first line of the project file identifies the source program:

```
_string[]   // program name should be "ContamW" (I1)
_string[]   // program version should be "2.0" (I1)
_echo       // (0/1) echo input files to the log file (I2)
```

The second line is a user-defined description of the project:

```
_prjdesc[]  // (unused) project description (I1) {W}
```

The next line sets the following miscellaneous values:

```
_skwidth    // total SketchPad width [cells] (I2) {W}
_skheight   // total SketchPad height [cells] (I2) {W}
_def_units  // default units: 0 = SI, 1 = US  (I2) {W}
_def_flows  // default flows: 0 = mass, 1 = volume  (I2) {W}
_def_T      // default temperature for zones [K] (R4) {W}
_udefT      // units for temperature  (I2) {W}
_rel_N      // angle to true north [degrees] (R4) {W}
_wind_H     // elevation for reference wind speed  [m] (R4) {W}
_uwH        // units for _wind_H  (I2) {W}
_wind_Ao    // local terrain constant  (R4) {W}
_wind_a     // velocity profile exponent  (R4) {W}
```

The next line defines the weather (WTHDAT) for steady-state simulation in ContamX:

```
Tambt       // ambient temperature [K] (R4)
barpres     // barometric pressure [Pa] NOT corrected to sea level (R4)
windspd     // wind speed [m/s] (R4)
winddir     // wind direction: 0 = N, 90 = E, 180 = S, ...; (R4)
relhum      // relative humidity: 0.0 to 1.0 (R4)
daytyp      // type of day (1-12) (I2)
uTa         // units for Tambt (I2) {W}
ubP         // units for barpres (I2) {W}
uws         // units for windspd (I2) {W}
uwd         // units for winddir (I2) {W}
```

The next line defines weather data (WTHDAT) for the wind pressure test in ContamW:

```
Tambt       // ambient temperature [K] (R4) {W}
barpres     // barometric pressure [Pa] NOT corrected to sea level (R4) {W}
windspd     // wind speed [m/s]  (R4) {W}
winddir     // wind direction: 0 = N, 90 = E, 180 = S, ...; (R4) {W}
relhum      // relative humidity: 0.0 to 1.0 (R4) {W}
daytyp      // type of day (1-12) (I2) {W}
uTa         // units for Tambt (I2) {W}
ubP         // units for barpres (I2) {W}
uws         // units for windspd (I2) {W}
uwd         // units for winddir (I2) {W}
```

The next two lines define the weather and contaminant files, respectively:

```
_WTHpath[_MAX_PATH]  // full name of weather file (I1)
_CTMpath[_MAX_PATH]  // full name of contaminant file (I1)
_CVFpath[_MAX_PATH]  // full name of continuous values file (I1) {Contam 2.1}
_DVFpath[_MAX_PATH]  // full name of discrete values file (I1) {Contam 2.1}
```

The following lines define path location data (PLDDAT) for the creating the WPC file:

{Contam 2.1}

```
_WPCfile[_MAX_PATH]  // full name of WPC file (I1)
EWCfile[_MAX_PATH]   // full name of EWC data source file (I1) {W}
WPCdesc[]   // WPC description (I1) {W}
X0          // X-value of ContamW origin in EWC coordinates [m] (R4) {W}
Y0          // Y-value of ContamW origin in EWC coordinates [m] (R4) {W}
Z0          // Z-value of ContamW origin in EWC coordinates [m] (R4) {W}
angle       // Rotation of ContamW relative to EWC coordinates (R4) {W}
u_XYZ       // units of coordinates (I2) {W}
epsPath     // tolerance for matching path locations [-] (R4) {W}
epsSpcs     // tolerance for matching species [-] (R4) {W}
tShift      // time shift of EWC data {W} [s] (hh:mm:ss → IX) {W}
dStart      // date WPC data starts (I2) {W}
dEnd        // date WPC data ends (I2) {W}
_useWPCwp   // if true, use WPC file wind pressures (I2)
_useWPCmf   // if true, use WPC file mass fractions (I2)
```

The next line defines the location (LOCDAT) (for future use with thermal simulation):

```
latd       // latitude (degrees: north +, south -) (R4)
lgtd       // longitude (degrees: east +, west -) (R4)
Tznr       // time zone (Greenwich = 0, Eastern = -5, etc.) (R4)
altd       // elevation above sea level [m] (R4)
Tgrnd      // ground temperature [K] (R4)
utg        // units for ground temperatures (I2)
u_a        // units for elevation (I2)
```

The remaining data is stored in the run control (RCDAT) structure. In ContamX some values may be transferred to other variables before being used. The next two lines control the airflow simulation – first the nonlinear part:

```
sim_af      // airflow simulation: 0 = steady, 1 = dynamic (I2)
afcalc      // N-R method for non-linear eqns: 0 = SUR, 1 = STR (I2)
afmaxi      // maximum number of N-R iterations (I2)
afrcnvg     // relative airflow convergence factor (R4)
afacnvg     // absolute airflow convergence factor [kg/s] (R4)
afrelax     // flow under-relaxation coefficient (for SUR) (R4)
uac2        // units for afacnvg (I2)
```

and then the linear part:

```
afslae      // method for linear equations: 0 = SKY, 1 = PCG (I2)
aflmaxi     // maximum number of iterations (PCG) (I2)
aflcnvg     // relative convergence factor for (PCG) (R4)
afrseq      // if true, resequence the linear equations (I2)
aflinit     // if true, do linear airflow initialization (I2)
Tadj        // if true, use temperature adjustment (I2)
```

The next three lines control the mass fraction calculation – first for cyclic simulation:

```
sim_mf      // mass fraction (contaminant) simulation:
            // 0 = none, 1 = steady, 2 = transient, 3 = cyclic (I2)
ccmaxi      // simulation: maximum number of cyclic iterations (I2)
ccrcnvg     // relative convergence factor (R4)
ccacnvg     // absolute convergence factor [kg/kg] (R4)
ccrelax     // (unused) over-relaxation coefficient (R4)
```

```
uccc        // units for ccacnvg (I2)
```

then for non-trace contaminants:

```
mfnmthd     // simulation: 0 = SKY, 1 = BCG, 2 = SOR (I2)
mfnmaxi     // maximum iterations (I2)
mfnrcnvg    // desired relative convergence (R4)
mfnacnvg    // desired absolute convergence (R4)
mfnrelax    // relaxation coefficient (R4)
mfngamma    // trapezoidal integration factor (R4)
uccn        // units for mfnacnvg (I2)
```

and then for trace contaminants:

```
mftmthd     // 0 = SKY, 1 = BCG, 2 = SOR (I2)
mftmaxi     // maximum iterations (I2)
mftrcnvg    // desired relative convergence (R4)
mftacnvg    // desired absolute convergence (R4)
mftrelax    // relaxation coefficient (R4)
mftgamma    // trapezoidal integration factor (R4)
ucct        // units for mftacnvg (I2)
```

The next line has four parameters; only the ones relating to density changes are active:

```
sim_vt      // (inactive) (0/1) variable zone temperature simulation (I2)
tsdens      // (0/1) vary density during time step (I2)
tsrelax     // (inactive) under-relaxation factor for calculating dM/dt (R4)
tsmaxi      // maximum number of iterations for density changes (I2)
```

The next line sets the dates, times, and time steps for simulation:

```
date_st     // day-of-year to start steady simulation (mmmdd → IX)
time_st     // time-of-day to start steady simulation (hh:mm:ss → I4)
date_0      // day-of-year to start transient simulation (mmmdd → IX)
time_0      // time-of-day to start transient simulation (hh:mm:ss → I4)
date_1      // day-of-year to end transient simulation (mmmdd → IX)
time_1      // time-of-day to end transient simulation (hh:mm:ss → I4)
time_step   // simulation time step [s] (hh:mm:ss → IX)
time_list   // listing (results) time step [s] (hh:mm:ss → IX)
time_scrn   // screen time step [s] (up to 1 day) (hh:mm:ss → I4)
```

The next line controls the use of the restart file:

```
restart     // use restart file (I2)
rstdate     // restart date (mmmdd → IX)
rsttime     // restart time (hh:mm:ss → I4)
```

The remaining parameters control the simulation outputs. They are in global variables in ContamX and the save[] vector in ContamW for flexibility. They may be spread over several lines:

```
_list       // data dump parameter (I2)
            //  > 0  dump matrix analysis,
            //  = 2  dump SIM file output,
            //  > 2  dump lognotes.
_pfsave / save[0]  // (0/1) save path flow results to SIM file (I1)
_zfsave / save[1]  // (0/1) save zone flow results to SIM file (I1)
_zcsave / save[2]  // (0/1) save mass fraction results to SIM file (I1)
_achvol / save[3]  // (0/1) ACH based on true volumes instead of std volumes
```

```
_achsave / save[4]  // (0/1) save building air exchange rate transient data
_abwsave / save[5]  // (0/1) save air exchange rate box-whisker data
_cbwsave / save[6]  // (0/1) save contaminant box-whisker data
_expsave / save[7]  // (0/1) save exposure transient data
_ebwsave / save[8]  // (0/1) save exposure box-whisker data
_zaasave / save[9]  // (0/1) save zones age-of-air transient data
_zbwsave / save[10] // (0/1) save zones age-of-air box-whisker data
save[11-30] // (unused; values 3–30 subject to change without notice) (I1)
save[31]    // ContamW will cause ContamX to display a dialog box (I1) {W}
```

The run control section is terminated with:

```
-999       // used to check for a read error in the above data
```

Note on outputs: _pfsave and _zfsave always have the same value – 0 or 1. The summaries, save[3] to save[10] are not currently active. They were developed for Contam96 to summarize the results of long simulations.

Example:

```
ContamW 2.1  0
Test CONTAMW for proper display of all elements.
! rows cols ud uf    T   uT     N     wH  u  Ao    a
    58   66  0  0 296.150 2    0.00 10.00 0 0.600 0.280
! Ta       Pb      Ws    Wd    rh  day u..
293.150 101325.0  0.000   0.0 0.500 1 2 0 0 1 ! steady simulation
293.150 101325.0  1.000 270.0 0.000 1 2 0 0 1 ! wind pressure test
null ! no weather file
null ! no contaminant file
null ! no continuous values file
null ! no discrete values file
null ! no WPC file
null ! no EWC file
WPC description
!  Xref    Yref    Zref   angle u
   0.000   0.000   0.000   0.00 0
! epsP epsS  tShift  dStart dEnd wp mf
  0.01 0.01 00:00:00   1/1   1/1  0  0
! latd  longtd   tznr  altd  Tgrnd u..
 40.00  -90.00  -6.00     0 283.15 2 0
!sim_af afcalc afmaxi afrcnvg afacnvg afrelax uac
     1      1     30  0.0001  1e-005    0.75   0
!   afslae aflmaxi aflcnvg afrseq aflinit Tadj
        0     100  1e-006      1       1    0
!sim_mf method maxi   relcnvg   abscnvg relax gamma ucc
     2           30 1.00e-004 1.00e-005 1.250         0 ! (cyclic)
            0   100 1.00e-006 1.00e-015 1.100 1.000   0 ! (non-trace)
            0   100 1.00e-006 1.00e-015 1.100 1.000   0 ! (trace)
!sim_vt tsdens relax tsmaxi
     0      0   0.75    20
!date_st time_st  date_0 time_0   date_1 time_1    t_step   t_list   t_scrn
  Jan01 00:00:00  Jan01 00:00:00  Jan01 24:00:00  01:00:00 01:00:00 01:00:00
!restart  date  time
      0  Jan01 00:00:00
!list pfsave zfsave zcsave
```

```
   0      1      1      1
!vol ach -bw cbw exp -bw age -bw
  0   0   0   0   0   0   0   0
!... _doDlg
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-999
```

❑ Section 2: Species and Contaminants

Species/contaminant data are read by the spcs_read( ) function and saved by the spcs_save( ) function in ContamX. They are read by ctm_read( ) in ContamX.

The species/contaminant section starts with:

```
_nctm      // total number of species simulated (= contaminants) (I2)
```

The next line lists the numbers of the _nctm species treated as contaminants. The ordering of the contaminants is important, and is set in ContamW.

```
_ctm[i]    // i = 0 to nctm-1
```

The third line of this section gives:

```
_nspcs     // the number of species
```

This is followed by a header line and then two lines of data for each species:

The first line of species data consists of:

```
nr         // species number (I2), in order from 1 to _nspcs
sflag      // 1 = simulated, 0 = unsimulated species (I2) {W}
ntflag     // 1 = non-trace, 0 = trace species (I2) {W}
molwt      // molecular weight - gas (R4)
mdiam      // mean diameter – particle [m] (R4)
edens      // effective densiity – particle [kg/m^3] (R4)
decay      // decay constant [1/s] (R4) {W}
ccdef      // default concentration [kg/kg air] (R4)
Cp         // (unused) specific heat at constant pressure [J/kgK] (R4)
ucc        // units to display concentration (I2) {W}
name[]     // species name (I1)
```

The second line is:

```
desc[]     // species description (I1) {W}
```

The section is terminated with:

```
-999       // used to check for a read error in the above data (I2)
```

Example:

```
2 ! contaminants:
   1 2
4 ! species:
! # s t  molwt   mdiam      edens      decay      CCdef       Cp       u...  name
  1 1 0  44.0000 0.000e+000 0.000e+000 0.000e+000 5.3184e-004 1000.000 1 0 0 C1
  2 1 0  44.0000 0.000e+000 0.000e+000 0.000e+000 6.8379e-004 1000.000 1 0 0 C2
  3 0 1  28.9645 0.000e+000 0.000e+000 0.000e+000 1.0000e+000    0.000 0 0 0 DryAir
Dry Air
  4 0 1  18.0153 0.0000e+000 0.0000e+000 0.0000e+000 5.0000e-003    0.000 0 0 0 H2O
Water Vapor
-999
```

❑ Section 3: Level and Icon Data

The data for recreating the SketchPad is stored in this section. The data are read by the level_read( ) function and saved by the level_save( ) function. The level data are stored in LEV_DATA structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The icon data are stored in ICON_DAT structures in ContamX and are not saved in ContamX.

The level/icons section starts with:

```
nlev       // number of levels (I2)
```

This is followed by a data header comment line and then data for all nlev levels.

Each level has a data line that includes:

```
   nr        // level number (I2), in order from 1 to nlev
   refht     // reference elevation of level [m] (R4)
   delht     // delta elevation to next level [m] (R4) {W}
   nicon     // number of icons on this level (I2)
   u_rfht    // units of reference elevation (I2) {W}
   u_dlht    // units of delta elevation (I2) {W}
   name[]    // level name (I1)
```

This line is followed by a comment line and then data for nicon icons.

Each icon has a data line consisting of:

```
   icon     // icon type – see 'special symbols' in contam.h (I2) {W}
   row      // row position on the SketchPad (I2) {W}
   col      // column position on the SketchPad (I2) {W}
   nr       // zone, path, duct, etc., number (I2) {W}
```

The section is terminated with:

```
-999       // used to check for a read error in the above data (I2)
```

Example:

```
2 ! levels plus icon data:
! #  refHt   delHt  ni  u  name
  1   0.000   3.000 44 0 0 one
!icn col row  #
  14  15  17   0
  23  19  17   5
  23  21  17   6
  23  23  17   7
  23  25  17   8
  23  27  17   9
...
  16  49  33   0
  42  19  34   6
  2   3.000   3.000 78 0 0 <3>
!icn col row  #
 162  10  14   1
 145  11  14   0
...
 185  32  38   7
 185  33  38   8
-999
```

❑ Section 4: Day Schedules

Day schedule data are read by the dschd_read( ) function and saved by the dschd_save( ) function. The data are stored in the DY_SCHD structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The day schedules section starts with:

```
_ndsch     // number of day schedules (I2)
```

This is followed by a data header comment line and then data for all _ndsch schedules.

For each schedule the first data line includes:

```
   nr        // schedule number (I2); in order from 1 to _ndsch
   npts      // number of data points (I2)
   shape     // 0 = rectangular; 1 = trapezoidal (I2)
   utyp      // type of units (I2) {W}
   ucnv      // units conversion (I2) {W}
   name[]    // schedule name (I1) {W}
```

and the second line has:

```
   desc[]    // schedule description (I1) {W} may be blank
```

This is followed by a line for each of the npts *data points:*

```
   time     // time-of-day [s] (hh:mm:ss converted to I4)
   ctrl     // corresponding control value (R4) [-]
```

The section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
1 ! day-schedules:
! # npts shap utyp ucnv name
  1    2    0    1    0 DySchd1
Always On
 00:00:00 1
 24:00:00 1
-999
```

❑ Section 5: Week Schedules

Week schedule data are read by the wschd_read( ) function and saved by the wschd_save( ) function. The data are stored in the WK_SCHD structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The week schedules section starts with:

```
_nwsch     // number of week schedules (I2)
```

This is followed by a data header comment line and then data for all _nwsch schedules.

For each week schedule the first data line includes:

```
   nr        // schedule number (I2); in order from 1 to _nwsch
   utyp      // type of units (I2) {W}
   ucnv      // units conversion (I2) {W}
   name[]    // schedule name (I1) {W}
```

and the second line has:

```
   desc[]    // schedule description (I1) {W} may be blank
```

and the third line has:

```
   j1 … j12  // 12 day schedule indices (I2) – converted to pointers
```

The section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
1 ! week-schedules:
! # utyp ucnv name
  1    1    0 WkSchd1
Week schedule.
 1 1 1 1 1 1 1 1 1 1 1 1
-999
```

❑ Section 6: Wind Pressure Profiles

Week pressure profile data are read by the wind_read( ) function and saved by the wind_save( ) function. The data are stored in the WIND_PF structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The wind pressure profiles section starts with:

```
_nwpf      // number of wind pressure profiles (I2)
```

This is followed by a data header comment line and then data for all _nwpf profiles.

For each wind pressure profile the first data line includes:

```
   nr         // profile number (I2); in order from 1 to _nwpf
   npts       // number of data points (I2)
   type       // 1 = linear; 2 = cubic spline; 3 = trigonometric (I2)
   name[]     // schedule name (I1) {W}
```

and the second line has:

```
   desc[]     // profile description (I1) {W} may be blank
```

This is followed by a line for each of the npts data points:

```
azm[]     // wind azimuth value {R4} [degrees]
coef[]    // normalized wind pressure coefficients {R4} [-]
```

The data points are used to compute the cubic spline or trigonometric coefficients.

The section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
2 ! wind pressure profiles:
1 5 1 WindProf1
Wind pressure profile one.
   0.0  1.000
  90.0  0.500
 180.0  1.000
 270.0  0.500
 360.0  1.000
2 9 1 WindProf2
Wind pressure profile two.
   0.0  1.000
  45.0  0.800
```

```
  90.0  0.200
 135.0  0.800
 180.0  1.000
 225.0  0.800
 270.0  0.200
 315.0  0.800
 360.0  1.000
-999
```

❑ Section 7: Kinetic Reactions

Kinetic reaction data are read by the kinetic_read( ) function and saved by the kinetic_save( ) function. The data are stored in the KNR_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The kinetic reactions section starts with:

```
_nkinr     // number of kinetic reactions (I2)
```

This is followed by a data header comment line and then data for all _nkinr reactions.

For each reaction the first data line includes:

```
   nr        // reaction number (I2); in order from 1 to _nkinr
   nkrd      // number of reactions (I2)
   name[]    // reaction name (I1) {W}
```

and the second line has:

```
   desc[]    // reaction description (I1) {W} may be blank
```

This is followed by nkrd lines of reaction data:

```
            prod[]   // product species name (I1)
            src[]    // source species name (I1)
            coef     // reaction coefficient (R4)
```

The kinetic reactions section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
1 ! kinetic reactions:
1 2 Kinr1
Kinetic reaction one.
 C1 C1 0.5
 C2 C2 0.5
-999
```

❑ Section 8: Filters

Filter data are read by the filter_read( ) function and saved by the filter_save( ) function. The data are stored in the FLT_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The filter section starts with:

```
_nfilt     // number of filters (I2)
```

This is followed by a data header comment line and then data for all _nfilt filters.

For each reaction the first data line includes:

```
   nr        // filter number (I2); in order from 1 to _nfilt
   nspcs     // number of species (I2)
   name[]    // filter name (I1) {W}
```

and the second line has:

```
   desc[]    // filter description (I1) {W} may be blank
```

This is followed by nspcs lines of filter efficiency data:

```
            spcs[]   // species name (I1)
            eff      // filter efficiency (R4)
```

The filter section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
1 ! filters:
1 2 Flt1

 C1 0.1
 C2 0.2
-999
```

❑ Section 9: Source/Sink Elements

Source/sink elements are read by the cselmt_read( ) function and saved by the cselmt_save( ) function. The data are stored in the CSE_DAT structures that are defined in contam.h for ContamW and simdat.h for ContamX. CSE_DAT includes a pointer to the element specific data structure.

The source/sink elements section starts with:

```
_ncse     // number of source/sink elements
```

This is followed by a data header comment line and then data for all _ncse elements.

For each element the first data line includes:

```
   nr        // element number (I2); in order from 1 to _ncse
   spcs[]    // species name (I1)
   ctype     // element data type (string → I2)
                   element type names are stored in _cse_dnames in ctype
order.
   name[]    // element name (I1) {W}
```

and the second line has:

```
   desc[]    // element description (I1) {W} may be blank
```

This is followed by one or more lines of data that depend on the element data type.

The introductory lines that follow give the type number, the program macro defined as that number, the type name from_cse_dnames, and the data structures to hold the data. The data structures are defined in celmts.h in ContamW and selmts.h in ContamX. Type numbers can change as long as the ordering in _csse_names and related arrays reflect that order.

Element type 0 [CS_CCF] "ccf" stored in structure CSE_CCF.

The constant coefficient source model data of:

```
   G         // generation rate [kg/s] (R4)
   D         // deposition rate [kg/s] (R4)
   u_G       // units of generation (I2) {W}
   u_D       // units of deposition (I2) {W}
```

Element type 1 [CS_PRS] "prs" stored in structure CSE_PRS.

The pressure driven source data of:

```
   G         // generation rate [kg/s] (R4)
   x         // pressure exponent [-] (R4)
   u_G       // units of generation (I2) {W}
```

Element type 2 [CS_CUT] "cut" stored in structure CSE_CUT.

The concentration cutoff model data of:

```
   G         // generation rate [kg/s] (R4)
   Co        // cutoff concentration [kg/kg] (R4)
   u_G       // units of generation (I2) {W}
   u_C       // units of concentration (I2) {W}
```

Element type 3 [CS_EDS] "eds" stored in structure CSE_EDS.

The exponential decay model data of:

```
   G0        // initial generation rate [kg/s] (R4)
   k         // decay constant [1/s] (R4)
   u_G       // units of generation (I2) {W}
   u_k       // units of time (I2) {W}
```

Element type 4 [CS_BLS] "bls" stored in structure CSE_BLS.

The boundary layer diffusion model data of:

```
   hm;       // average film mass transfer coefficient [m/s] (R4)
   rho;      // film density of air [kg/m^3] (R4)
   Kp;       // partition coefficient (R4)
   M;        // adsorbent mass/unit area [kg/m^2] (R4)
   u_h;      // units of hm (I2) {W}
   u_r;      // units of rho (I2) {W}
   u_M;      // units of M (I2) {W}
```

Element type 5 [CS_BRS] "brs" stored in structure CSE_BRS.

The burst source data of:

```
   M         // mass added to zone in one time step [kg] (R4)
   u_M       // units of mass (I2) {W}
```

The source/sink element section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
6 ! source/sink elements:
1 C1 ccf s1
```

```
Constant Coefficient
 1 1 0 0
2 C1 prs s2
Pressure driven
 2 2 0
3 C1 cut s3
Cut-off Concentration
 3 3 0 0
4 C1 eds s4
Decaying Source
 4 4 0 0
5 C1 bls s5
Boundary Layer Diffusion
 5 5 5 5 0 0 0
6 C2 brs s6
Burst
 6 0
-999
```

❑ Section 10: Airflow Elements

Airflow elements are read by the afelmt_read( ) function and saved by the afelmt_save( ) function. The data are stored in the AFE_DAT structures that are defined in contam.h for ContamW and simdat.h for ContamX. AFE_DAT includes a pointer to the element specific data structure.

The airflow elements section starts with:

```
_nafe     // number of airflow elements
```

This is followed by a data header comment line and then data for all _nafe elements.

For each element the first data line includes:

```
   nr        // element number (I2); in order from 1 to _nafe
   icon      // icon used to represent flow path (R4) {W}
   dtype     // element data type (string → I2)
             // element type names are stored in _afe_dnames in dtype order.
   name[]    // element name (I1) {W}
```

and the second line has:

```
   desc[]    // element description (I1) {W} may be blank
```

This is followed by one or more lines of data that depend on the element data type.

The introductory lines that follow give the type number, the program macro defined as that number, the type name from _afe_dnames, and the data structures to hold the data. The data structures are defined in celmts.h in ContamW and selmts.h in ContamX. Type numbers can change as long as the ordering in _afe_dnames and related arrays reflect that order.

Element type 0 [PL_ORFC] "plr_orfc" stored in structure PLR_ORF.

The orifice data consist of:

```
   lam       // laminar flow coefficient (R4)
   turb      // turbulent flow coefficient (R4)
   expt      // pressure exponent (R4)
```

```
area      // actual area [m^2] (R4) {X}
dia       // hydraulic diameter [m] (R4) {X}
coef      // flow coefficient (R4) {X}
Re        // laminar/turbulet transition Reynolds number [-](R4) {X}
u_A       // units of area (I2) {X}
u_D       // units of diameter (I2) {X}
```

Element type 1 [PL_LEAK1] "plr_leak1" stored in structure PLR_LEAK.

Element type 2 [PL_LEAK2] "plr_leak2" stored in structure PLR_LEAK.

Element type 3 [PL_LEAK3] "plr_leak3" stored in structure PLR_LEAK.

The leakage area data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
coef      // flow coefficient (R4) {W}
pres      // reference pressure drop [Pa] (R4) {W}
area1     // leakage area per item [m^2] (R4) {W}
area2     // leakage area per unit length [m^2/m] (R4) {W}
area3     // leakage area per unit area [m^2/m^2] (R4) {W}
u_A1      // units of area1 [m^2] (I2) {W}
u_A2      // units of area2 [m^2/m] (I2) {W}
u_A3      // units of area3 [m^2/m^2] (I2) {W}
u_dP      // units of pressure (I2) {W}
```

Element type 4 [PL_CONN] "plr_conn" stored in structure PLR_CONN.

The (ASCOS compatible) connection data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent - 0.5 (R4)
area      // actual area [m^2] (R4) {W}
coef      // flow coefficient (R4) {W}
u_A       // units of area (I2) {W}
```

Element type 5 [PL_QCN] "plr_qcn" stored in structure PLR_QCN.

The volume flow powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
```

Element type 6 [PL_FCN] "plr_fcn" stored in structure PLR_FCN.

The mass flow powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
```

Element type 7 [PL_TEST1] "plr_test1" stored in structure PLR_TEST1.

The single test point powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
dP        // pressure drop [Pa] (R4) {W}
Flow      // flow rate [kg/s] (R4) {W}
u_P       // units of pressure drop  {W}
u_F       // units of flow (I2) {W}
```

Element type 8 [PL_TEST2] "plr_test2" stored in structure PLR_TEST2.

The two test points powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
dP1       // point 1 pressure drop [Pa] (R4) {W}
F1        // point 1 flow rate [kg/s] (R4) {W}
dP2       // point 2 pressure drop [Pa] (R4) {W}
F2        // point 2 flow rate [kg/s] (R4) {W}
u_P1      // units of pressure drop (I2) {W}
u_F1      // units of flow (I2) {W}
u_P2      // units of pressure drop (I2) {W}
u_F2      // units of flow (I2) {W}
```

Element type 9 [PL_CRACK] "plr_crack" stored in structure PLR_CRACK.

The crack powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
length    // crack length [m] (R4) {W}
width     // crack width [m] (R4) {W}
u_L       // units of length (I2) {W}
u_W       // units of width (I2) {W}
```

Element type 10 [PL_STAIR] "plr_stair" stored in structure PLR_STAIR.

The stairwell powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
Ht        // distance between levels [m] (R4) {W}
Area      // cross-sectional area [m^2] (R4) {W}
peo       // density of people [pers/m^2] (R4) {W}
tread     // 1 = open tread 0 = closed  {W}
u_A       // units of area (I2) {W}
u_D       // units of distance (I2) {W}
```

Element type 11 [PL_SHAFT] "plr_shaft" stored in structure PLR_SHAFT.

The shaft powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
Ht        // distance between levels [m] (R4) {W}
Area      // cross-sectional area [m^2] (R4) {W}
Perim     // perimeter [m] (R4) {W}
rough     // roughness [m] (R4) {W}
u_A       // units of area (I2) {W}
u_D       // units of distance (I2) {W}
u_P       // units of perimeter (I2) {W}
u_R       // units of roughness (I2) {W}
```

Element type 12 [PL_BDQ] "plr_bdq" stored in structure PLR_BDQ.

The volume flow powerlaw backdraft damper data consist of:

```
lam       // laminar flow coefficient {R4}
Cp        // turbulent flow coefficient ( dP > 0 ) {R4}
xp        // pressure exponent ( dP > 0 ) {R4}
Cn        // turbulent flow coefficient ( dP < 0 ) {R4}
xn        // pressure exponent ( dP < 0 ) {R4}
```

Element type 13 [PL_BDF] "plr_bdf" stored in structure PLR_BDF.

The mass flow powerlaw backdraft damper data consist of:

```
lam       // laminar flow coefficient {R4}
Cp        // turbulent flow coefficient ( dP > 0 ) {R4}
xp        // pressure exponent ( dP > 0 ) {R4}
Cn        // turbulent flow coefficient ( dP < 0 ) {R4}
xn        // pressure exponent ( dP < 0 ) {R4}
```

Element type 14 [QFR_QAB] "qfr_qab" stored in structure QFR_QAB.

The volume flow quadratic data consist of:

```
a         // dP = a*Q + b*Q*Q {R4}
b         // {R4}
```

Element type 15 [QFR_QAF] "qfr_fab" stored in structure QFR_FAB.

The mass flow quadratic data consist of:

```
a         // dP = a*F + b*F*F {R4}
b         // {R4}
```

Element type 16 [QFR_CRACK] "qfr_crack" stored in structure QFR_CRACK.

The crack mass flow quadratic data consist of:

```
a         // dP = a*F + b*F*F {R4}
b         // {R4}
length    // crack length [m] {R4}
width     // crack width [m] {R4}
depth     // crack depth [m] {R4}
```

```
nB        // number of bends
u_L       // units of length
u_W       // units of width
u_D       // units of depth
```

Element type 17 [QFR_TEST2] "qfr_test2" stored in structure QFR_TEST2.

The two test points mass flow quadratic data consist of:

```
a         // dP = a*F + b*F*F {R4}
b         // {R4}
dP1       // point 1 pressure drop [Pa] (R4) {W}
F1        // point 1 flow rate [kg/s] (R4) {W}
dP2       // point 2 pressure drop [Pa] (R4) {W}
F2        // point 2 flow rate [kg/s] (R4) {W}
u_P1      // units of pressure drop (I2) {W}
u_F1      // units of flow (I2) {W}
u_P2      // units of pressure drop (I2) {W}
u_F2      // units of flow (I2) {W}
```

Element type 18 [DR_DOOR] "dor_door" stored in structure AFE_DOR.

The single opening doorway data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
dTmin     // minimum temperature difference for two-way flow [C] (R4)
ht        // height of doorway [m] (R4)
wd        // width of doorway [m] (R4)
cd        // discharge coefficient
u_T       // units of temperature (I2) {W}
u_H       // units of height (I2) {W}
u_W       // units of width (I2) {W}
```

Element type 19 [DR_PL2] "dor_pl2" stored in structure DR_PL2.

The double opening doorway data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
dH        // distance above | below midpoint [m] {R4}
ht        // height of doorway [m] (R4) {W in v. 2.0}
wd        // width of doorway [m] (R4) {W}
cd        // discharge coefficient [-] (R4) {W}
u_H       // units of height (I2) {W}
u_W       // units of width (I2) {W}
```

Element type 20 [FN_CMF] "fan_cmf" stored in structure AFE_CMF.

The constant mass flow fan data consist of:

```
Flow      // design flow rate [kg/s] (R4)
u_F       // units of flow (I2) {W}
```

Element type 21 [FN_CVF] "fan_cvf" stored in structure AFE_CVF.

The constant volume flow fan data consist of:

```
Flow      // design flow rate [m^3/s] (R4)
u_F       // units of flow (I2) {W}
```

Element type 22 [FN_FAN] "fan_fan" stored in structure AFE_FAN.

The first line of performance curve fan data consists of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
rdens     // reference fluid density [kg/m^3] {R4}
fdf       // free delivery flow (prise = 0) [kg/s] {R4}
sop       // shut-off pressure (flow = 0) [Pa] {R4}
off       // fan is off if (RPM/rated RPM) < off {R4}
```

The second line consists of:

```
fpc[4]    // fan performance polynomial coefficients {R4}
npts      // number of mesaured data points (I2) {W}
Sarea     // shut-off orifice area [m^2] {R4} {W}
u_Sa      // units of shut-off area (I2) {W}
The next npts lines consists of:
mF        // measured flow rates [kg/s] (R4) {W}
u_mF      // units of measured flows (I2) {W}
dP        // measured pressure rises [Pa] (R4) {W}
u_dP      // units of pressure rises (I2) {W}
rP        // revised pressure rises [Pa] (R4) {W}
u_rP      // units of revised pressures (I2) {W}
```

The airflow element section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
24 ! flow|duct elements:
1 24 dor_door DR_DOOR
DR_DOOR Single opening w/ 2-way flow
 0.0741669 1.76494 0.5 0.01 2 0.8 0.78 0 0 0
2 24 dor_pl2 DR_PL2
DR_PL2 Two-opening model
 0.0185417 0.882469 0.5 0.444444 2 0.8 0.78 0 0
3 30 fan_cmf FN_CMF
FN_CMF Constant mass flow fan model
 1 0
4 29 fan_cvf FN_CVF
FN_CVF Constant volume flow fan model
 1 0
5 30 fan_fan FN_FAN
FN_FAN Cubic polynomial fan model
 7.2e-006 0.00848528 0.5 1.2041 -0.377789 0.0286444 0.1
```

```
 0.0286444 0.0745766 -0.00327397 5.29862e-005 4 0.01 0
  1 0 0.1 0 0.1 0
  10 0 0.5 0 0.5 0
  30 0 0.75 0 0.75 0
  35 0 0.9 0 0.9 0
6 25 plr_conn PLR_CONN
ASCOS Connection element. Analysis of Smoke Control of Systems ...
 8.38053e-008 0.000188562 0.5 0.0002 0.666667 3
7 23 plr_crack PLR_CRACK
Crack element using powerlaw relationship.
 1.84e-008 0.000861959 0.68394 2 0.002 0 4
8 23 plr_fcn PLR_FCN
Power law for mass flow.
 3.52946e-005 0.01 0.5
9 23 plr_leak1 PLR_LEAK1
Leakage element (per item).
 2.4e-008 8.48528e-005 0.5 0.6 10 0.0001 0 0 2 2 2 0
10 23 plr_leak2 PLR_LEAK2
Leakage element (per unit length).
 3.8063e-009 6.00712e-005 0.65 0.6 10 0 0.0001 0 2 2 2 0
11 23 plr_leak3 PLR_LEAK3
Leakage element (per unit area).
 3.8063e-009 6.00712e-005 0.65 0.6 10 0 0 0.0001 2 2 2 1
12 23 plr_orfc PLR_ORFC
Orifice element.
 2.4e-005 0.00848528 0.5 0.01 0.1 0.6 30 0 0
13 23 plr_qcn PLR_QCN
Power law for volume flow.
 3.52946e-005 0.01 0.5
14 23 plr_shaft PLR_SHAFT
Shaft powerlaw model.
 0.306565 30.9005 0.508447 3 6 10 0.1 0 0 0 0
15 23 plr_stair PLR_STAIR
Stairwell powerlaw model.
 0.146333 2.83196 0.5 3 12.5 0 1 0 0
16 23 plr_test1 PLR_TEST1
Single point test data element.
 2.47343e-005 0.0086575 0.5 4 0.019 0 0
17 23 plr_test2 PLR_TEST2
Two point test data element.
 6.35906e-005 0.00913228 0.461488 4 0.019 10 0.029 0 0 0 0
18 23 plr_bdf PL_BDF
PL_BDF Backdraft Damper based on mass flow rate
 3.52946e-005 0.01 0.5 0.0001 0.5
19 23 plr_bdq PL_BDQ
PL_BDQ Backdraft Damper based on volume flow rate
 3.52946e-005 0.01 0.5 0.0001 0.5
20 23 qfr_crack QF_CRACK
This description is the maximum allowable length that CONTAMW allo
 565.645 90835.5 2 0.002 0.05 2 0 4 2
21 23 qfr_fab QF_FAB
QF_FAB Quadratic mass flow model
```

```
 1234.12 123456
22 23 qfr_qab QF_QAB
QF_QAB Qadratic volume flow model
 1 1234.12
23 23 qfr_test2 QF_TEST2
QF_TEST2 Quadratic test data (2 points)
 -44.6461 13430.1 4 0.019 10 0.029 0 0 0 0
24 30 fan_cmf a

 1 0
-999
```

❑ Section 11: Duct Elements

Duct elements are read by the afelmt_read( ) function with the duct flag set to 1 and saved by the afelmt_save( ) function. The data are stored in the AFE_DAT structures that are defined in contam.h for ContamW and simdat.h for ContamX. AFE_DAT includes a pointer to the element specific data structure.

The duct elements section starts with:

```
_ndfe     // number of duct flow elements
```

This is followed by a data header comment line and then data for all _ndfe elements.

For each element the first data line includes:

```
   nr        // element number (I2); in order from 1 to _nafe
   icon      // icon used to represent flow path ((R4) {W}
   dtype     // element data type (string → I2)
             // element type names are stored in _afe_dnames in dtype order.
   name[]    // element name (I1) {W}
```

and the second line has:

```
   desc[]    // element description (I1) {W} may be blank
```

This is followed by two or more lines of data that depend on the element data type.

The introductory lines that follow give the type number, the program macro defined as that number, the type name from _afe_dnames, and the data structures to hold the data. The data structures are defined in celmts.h in ContamW and selmts.h in ContamX. Type numbers can change as long as the ordering in _afe_dnames and related arrays reflect that order.

Element type 23 [DD_DWC] "dct_dwc" stored in structure DEF_DWC.

The Darcy-Colebrook data consist of:

```
   hdia      // hydraulic diameter [m] (R4)
   area      // cross sectional area [m^2] (R4)
   ed        // relative roughness (rough/hdia) (R4)
   lam       // laminar total loss coefficient per [m] of duct (R4)
   rough     // roughness dimension [m] (R4)
   u_R       // units for roughness (I2) {W}
```

Element type 24 [DD_PLR] "dct_plr" stored in structure DFE_ORF.

The orifice data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
area      // actual area [m^2] (R4) {X}
dia       // hydraulic diameter [m] (R4) {X}
coef      // flow coefficient (R4) {X}
Re        // laminar/turbulet transition Reynolds number [-](R4) {X}
u_A       // units of area (I2) {X}
u_D       // units of diameter (I2) {X}
```

Element type 26 [DD_QCN] "dct_qcn" stored in structure DFE_QCN.

The volume flow powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
```

Element type 25 [DD_FCN] "dct_fcn" stored in structure DFE_FCN.

The mass flow powerlaw data consist of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
```

Element type 28 [DD_CMF] "dct_cmf" stored in structure DFE_CMF.

The constant mass flow fan data consist of:

```
Flow      // design flow rate [kg/s] (R4)
u_F       // units of flow (I2) {W}
```

Element type 29 [29_CVF] "dct_cvf" stored in structure DFE_CVF.

The constant volume flow fan data consist of:

```
Flow      // design flow rate [m^3/s] (R4)
u_F       // units of flow (I2) {W}
```

Element type 27 [DD_FAN] "dct_fan" stored in structure DFE_FAN.

The first line of performance curve fan data consists of:

```
lam       // laminar flow coefficient (R4)
turb      // turbulent flow coefficient (R4)
expt      // pressure exponent (R4)
rdens     // reference fluid density [kg/m^3] {R4}
fdf       // free delivery flow (prise = 0) [kg/s] {R4}
sop       // shut-off pressure (flow = 0) [Pa] {R4}
off       // fan is off if (RPM/rated RPM) < off {R4}
```

The second line consists of:

```
fpc[4]    // fan performance polynomial coefficients {R4}
npts      // number of mesaured data points (I2) {W}
```

```
Sarea     // shut-off orifice area [m^2] {R4} {W}
u_Sa      // units of shut-off area (I2) {W}
```

The next npts lines consists of:

```
mF        // measured flow rates [kg/s] (R4) {W}
u_mF      // units of measured flows (I2) {W}
dP        // measured pressure rises [Pa] (R4) {W}
u_dP      // units of pressure rises (I2) {W}
rP        // revised pressure rises [Pa] (R4) {W}
u_rP      // units of revised pressures (I2) {W}
```

Element type 30 [DD_BDQ] "dct_bdq" stored in structure DFE_BDQ.

The volume flow powerlaw backdraft damper data consist of:

```
lam       // laminar flow coefficient {R4}
Cp        // turbulent flow coefficient ( dP > 0 ) {R4}
xp        // pressure exponent ( dP > 0 ) {R4}
Cn        // turbulent flow coefficient ( dP < 0 ) {R4}
xn        // pressure exponent ( dP < 0 ) {R4}
```

Element type 31 [DD_BDF] "dct_bdf" stored in structure DFE_BDF.

The mass flow powerlaw backdraft damper data consist of:

```
lam       // laminar flow coefficient {R4}
Cp        // turbulent flow coefficient ( dP > 0 ) {R4}
xp        // pressure exponent ( dP > 0 ) {R4}
Cn        // turbulent flow coefficient ( dP < 0 ) {R4}
xn        // pressure exponent ( dP < 0 ) {R4}
```

Each duct element then has the following geometry and leakage data stored in structure DUCT.

```
Hdia      // hydraulic diameter [m] {R4}
perim     // perimeter [m] {R4}
area      // cross sectional area [m^2] {R4}
major     // major dimension of rectangular or oval duct [m] {R4}
minor     // minor dimension of rectangular or oval duct [m] {R4}
As        // duct segment surface area [m^2] {R4}
Qr        // duct leakage rate at Pr [m^3/m^2] {R4}
Pr        // dPstatic for leakage rate [Pa] {R4}
CL        // ASHRAE leakage class number {R4}
Prs       // standard dP for leakage rate [Pa] {R4}
shape     // 0 = circle, 1 = rectangle, 2 = oval, 3 = other (I2) {W}
u_D       // units for diameter (I2) {W}
u_P       // units for perimeter (I2) {W}
u_A       // units for area (I2) {W}
u_mj      // units for major dimension (I2) {W}
u_mn      // units for minor dimension (I2) {W}
u_Qr      // units for leakage rate (I2) {W}
u_Pr      // units for standard dP (I2) {W}
```

The duct element section is terminated with:

```
-999      // used to check for a read error in the above data
```

Example:

```
9 ! flow|duct elements:
1 23 dct_dwc Dct1
Duct one.
 9e-005 0 4
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
2 23 dct_plr Dct2
Duct two
 2.4e-005 0.00848528 0.5 0.01 0.1 0.6 0 0
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
3 23 dct_fcn Dct3
Duct three.
 3.52946e-005 0.01 0.5
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
4 23 dct_qcn Dct4
Duct four
 3.52946e-005 0.01 0.5
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
5 23 dct_fan Dct5
Duct five
 7.2e-006 0.00848528 0.5 1.2041 4.53711 764.429 0.1
 764.429 -22.0238 28.2143 -13.3333 5 0.01 0
  0 0 765 0 765 0
  1 0 755 0 755 0
  2 0 730 0 730 0
  3 0 590 0 590 0
  4 0 275 0 275 0
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
6 23 dct_cmf Dct6
Duct 6
 1 0
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
7 23 dct_cvf Dct7
Duct seven
 1 0
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
8 23 dct_bdq Dct8
Duct eight
 3.52946e-005 0.01 0.5 0.0001 0.5
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
 0 4 4 2 4 4 0 0
9 23 dct_bdf Dct9
Duct nine
 3.52946e-005 0.01 0.5 0.0001 0.5
 0.2 0.628319 0.0314159 0.2 0.2 0 0 250
```

```
 0 4 4 2 4 4 0 0
-999
```

❑ Section 12: Control Nodes

Control node data are read by the ctrl_read( ) function and saved by the ctrl_save( ) function. The data are stored in the CTRL_DSC structures that are defined in contam.h for ContamW and the CT_NODE structures defined simdat.h for ContamX.

The control nodes section starts with:

```
_nctrl     // number of control nodes (I2)
```

This is followed by a data header comment line and then data for all _nctrl control nodes.

For each control node the first data line includes:

```
   nr        // node (SketchPad) number (I2); in order from 1 to _nzone
   type      // node data type (string → I2)
                Node type names are stored in _ctrl_names in type order.
   seqnr     // computation sequence number (I2); set in ContamW
   flags     // flags for offset & scale and time constant (U2)
   n1        // SketchPad number of input node #1 (I2)
   n2        // SketchPad number of input node #2 (I2)
   name[]    // element name (I1) {W}
```

and the second line has:

```
   desc[]    // control node description (I1) {W} may be blank
```

This may be followed by one more line of data that depends on the node type.

Node type 0 [CT_SNS] "sns" stored in structure SENSOR {W} or SNSDAT {X}.

```
   offset    // offset value (R4)
   scale     // scale value (R4)
   tau       // time constant (R4)
   oldsig    // signal at last time step - for restart (R4)
   source    // index of source (source not defined at read time) (I2)
   type      // type of source: 1=zone, 2=path, 3=junction, 4=duct... (I2)
   measure   // 0=contaminant, 1=temperature, 2=flow rate, 3=dP ... (I2)
   species[] // species name [I1]; convert to pointer
```

Node type 1 [CT_SCH] "sch" stored in structure SCHDAT.

```
ps        // week schedule index (I2); converted to pointer
```

Node type 2 [CT_SET] "set" stored in structure SETDAT.

```
value     // constant value (R4)
```

Node type 3 [CT_CVF] "cvf" stored in structure CDVDAT. {Contam 2.1}

```
Name[]    // name of the value read from the Continuous Values file (I1)
```

Node type 4 [CT_DVF] "dvf" stored in structure CDVDAT. {Contam 2.1}

```
Name[]    // name of the value read from the Discrete Values file (I1)
```

Node type 5 [CT_LOG] "log" stored in structure LOGDAT.

```
Offset    // offset value (R4)
Scale     // scale value (R4)
header[]  // header string (I1)
units[]   // units string (I1)
```

```
udef      // true if using default units (I2) {W}
```

Node type 6 [CT_PAS] "pas" has no additional data.

Node type 7 [CT_MOD] "mod" stored in structure MOD.

```
offset    // offset value (R4)
scale     // scale value (R4)
```

Node type 8 [CT_HYS] "hys" stored in structure HYSDAT.

```
slack;    // hysteresis parameter (R4)
slope     // 1.0 / (1.0 - slack) (R4)
oldsig    // prior output signal (R4)
```

Node type 9 [CT_ABS] "abs" has no additional data.

Node type 10 [CT_BIN] "bin" has no additional data.

Node type 11 [CT_DLS] "dls" stored in structure CDVDAT. {Contam 2.1}

```
dsincr   // day schedule number for increasing signal (I2)
dsdecr   // day schedule number for decreasing signal (I2)
```

Node type 12 [CT_DLX] "dlx" stored in structure CDVDAT. {Contam 2.1}

```
tauincr  // time constant for increasing signal [s] (I4)
taudecr  // time constant for decreasing signal [s] (I4)
```

Node type 13 [CT_INT] "int" has no additional data.

Node type 14 [CT_RAV] "rav" stored in structure CDVDAT. {Contam 2.1}

```
tspan    // time span for the running average [s] (I4)
```

Node type 15 [CT_INV] "inv" has no additional data.

Node type 16 [CT_AND] "and" has no additional data.

Node type 17 [CT_OR] "od" has no additional data.

Node type 18 [CT_XOR] "xor" has no additional data.

Node type 19 [CT_ADD] "add" has no additional data.

Node type 20 [CT_SUB] "sub" has no additional data.

Node type 21 [CT_MUL] "mul" has no additional data.

Node type 22 [CT_DIV] "div" has no additional data.

Node type 23 [CT_SUM] "sum" stored in structure SUMDAT.

Node type 24 [CT_AVG] "avg" stored in structure SUMDAT.

Node type 25 [CT_MAX] "max" stored in structure SUMDAT.

Node type 26 [CT_MIN] "min" stored in structure SUMDAT.

```
nval     // number of values to be processed (I2)
pc ...   // indices of nval control nodes (I2)
```

Node type 27 [CT_LLS] "lls" has no additional data.

Node type 28 [CT_ULS] "uls" has no additional data.

Node type 29 [CT_SUM] "sum" stored in structure BANDAT.

Node type 30 [CT_AVG] "avg" stored in structure BANDAT.

```
band     // width of bans (R4)
```

Node type 31 [CT_LLC] "llc" has no additional data.

Node type 32 [CT_ULC] "ulc" has no additional data.

Node type 33 [CT_PC1] "pc1" stored in structure PCDAT.

```
kp       // proportional gain factor (R4)
```

Node type 34 [CT_PIC] "pic" stored in structure PICDAT.

```
kp       // proportional gain factor (R4)
ki       // integral gain factor (R4)
oldsig   // prior output signal - for restart (R4)
olderr   // prior error value - for restart (R4)
```

The control nodes section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
8 ! control nodes:
! # typ seq f n  c1  c2  s# data
  1 sns   1 0 0   0   0 <none>
zone sensor
 0 1 0 0 1 1 0 C1
  2 sns   2 0 0   0   0 <none>
zone sensor
 0 1 0 0 1 1 0 C2
  3 log   5 0 1   2   0 Z5C2
report
 0 1 1 MassFraction kg/kg
  4 log   6 0 1   1   0 Z5C1
report
 0 1 1 MassFraction kg/kg
  5 sns   3 0 0   0   0 <none>
path sensor
 0 1 0 0 3 2 2 none
  6 sns   4 0 0   0   0 <none>
path sensor
 0 1 0 0 4 2 2 none
  7 log   7 0 1   5   0 Path-L
report
 0 1 1 FlowRate kg/s
  8 log   8 0 1   6   0 Path-R
report
 0 1 1 FlowRate kg/s
-999
```

❑ Section 13: Simple Air Handling System (AHS)

AHS data are read by the system_read( ) function and saved by the system_save( ) function. The data are stored in the AHS_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The AHS section starts with:

```
_nahs      // number of AHS (I2)
```

This is followed by a data header comment line and then data for all _nahs systems.

For each AHS the first data line includes:

```
   nr        // AHS number (I2); in order from 1 to _nahs
   zone_r    // return zone number (I2)
   zone_s    // supply zone number (I2)
   path_r    // recircultaion air path number (I2)
   path_s    // outdoor air path number (I2)
   path_x    // exhaust path number (I2)
```

and the second line has:

```
   desc[]    // AHS description (I1) {W} may be blank
```

The AHS section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
1 ! simple AHS:
! # zr# zs# pr# ps# px# name
  1   2   3  29  30  31 Ahs1
Simple Air Handling System #1
-999
```

❑ Section 14: Zones

Zone data are read by the zone_read( ) function and saved by the zone_save( ) function. The data are stored in the ZONE_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The zone section starts with:

```
_nzone     // number of zones (I2)
```

This is followed by a data header comment line and then data for all _nzone zones.

For each zone the data line includes:

```
   nr        // zone number (I2); in order from 1 to _nzone
   flags     // zone flags – bits defined in contam.h (U2)
   ps        // week schedule index (I2); converted to pointer
   pc        // control node index (I2); converted to pointer
   pk        // kinetic reaction index (I2); converted to pointer
   pl        // building level index (I2); converted to pointer
   relHt     // zone height [m] (R4)
   Vol       // zone volume [m^3] (R4)
   T0        // initial zone temperature [K] (R4)
   P0        // initial zone pressure [Pa] (R4)
   name[]    // zone name (I1) {W}
   u_Ht      // units of height (I2) {W}
   u_V       // units of volume (I2) {W}
   u_T       // units of temperature (I2) {W}
   u_P       // units of pressure (I2) {W}
```

The zones section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
7 ! zones:
! Z#  f  s#  c#  k#  l#  relHt    Vol   T0       P0  name u..
   1  3   0   0   0   2   0.000 5.00001 296.15    0  z5   0 2 0 0
   2 10   0   0   0   2   0.000 2       296.15    0  ahsR 0 2 0 0
   3 10   0   0   1   2   0.000 1       296.15    0  ahsS 0 2 0 0
   4  3   0   0   0   1   0.000 0.99999 296.15    0  z1   0 2 0 0
   5  3   0   0   0   1   0.000 2       293.15    0  z2   0 2 0 0
   6  3   0   0   0   1   0.000 3       296.15    0  z3   0 2 0 0
   7  3   0   0   0   1   0.000 4       296.15    0  z4   0 2 0 0
-999
```

❑ Section 15: Initial Zone Concentrations

Initial zone contaminant mass fractions are read by the zone_read( ) function and saved by the zone_save( ) function. The mass fractions are stored in the ZONE_DSC structures that are defined in contam.h for ContamW and they are stored in an array in ContamX.

The zone contaminants section starts with:

```
nn         // number of mass fraction that follow (I4)
```

*nn should equal _nzone * _nctm.*

This is followed by _nzone lines in order from 1 to _nzone.

Each data line contains _nctm mass fractions (R4) in contaminant order.

```
   nr
   CC0[i][1]      // initial mass fraction (R4) of zone i, contaminant 1
   ...
   CC0[i][_nctm]  // initial mass fraction (R4) of zone i, contaminant _nctm
```

The initial zone concentrations section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
14 ! initial concentrations:
! Z#         C1         C2
   1 1.000e+000 2.000e+000
   2 1.000e+000 2.000e+000
   3 1.000e+000 2.000e+000
   4 1.000e+000 2.000e+000
   5 1.000e+000 2.000e+000
   6 1.000e+000 2.000e+000
   7 1.000e+000 2.000e+000
-999
```

❑ Section 16: Airflow Paths

Path data are read by the path_read( ) function and saved by the path_save( ) function. The data are stored in the PATH_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The path section starts with:

```
_npath     // number of paths (I2)
```

This is followed by a data header comment line and then data for all _npath paths.

For each path the data line includes:

```
   nr        // path number (I2); in order from 1 to _npath
   flags     // airflow path flag values (I2)
   pzn       // zone N index (I2); converted to pointer
   pzm       // zone M index (I2); converted to pointer
   pe        // flow element index (I2); converted to pointer
   pf        // filter index (I2); converted to pointer
   pw        // wind coefficients index (I2); converted to pointer
   pa        // AHS index (I2); converted to pointer
   ps        // schedule index (I2); converted to pointer
   pc        // control node index (I2); converted to pointer
   pld       // level index (I2); converted to pointer
   X         // X-coordinate of envelope path [m] (R4) {Contam 2.1}
   Y         // Y-coordinate of envelope path [m] (R4) {Contam 2.1}
   relHt     // height relative to current level [m] (R4)
   mult      // element multiplier (R4)
   wPset     // constant wind pressure [Pa] (pw==NULL) (R4)
   wPmod     // wind speed(?) modifier (pw!=NULL) (R4)
   wazm      // wall azimuth angle (pw!=NULL) (R4)
   Fahs      // AHS path flow rate [kg/s] (pw==NULL) (R4)
   Xmax      // flow or pressure limit - maximum (R4) {W}
   Xmin      // flow or pressure limit - minimum (R4) {W}
   icon      // icon used to represent flow path (U1) {W}
   dir       // positive flow direction on sketchpad (U1) {W}
   u_Ht      // units of height (I2) {W}
   u_dP      // units of pressure difference (I2) {W}
   u_F       // units of flow (I2) {W}
```

The paths section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
31 ! flow paths:
! P# f  n# m# e# f# w# a# s# c# l#    X      Y   relHt  mult wPset wPmod wazm Fahs..
   1  8  1  2  0  0  0  1  0  0  2  0.000  0.000 0.000 1 0 0      0  2 0 0 129  5 0 0 0 0
   2  8  3  1  0  0  0  1  0  0  2  0.000  0.000 0.000 1 0 0      0  1 0 0 128  2 0 0 0 0
   3  7 -1  1 12  0  1  0  0  0  2  0.000  0.000 1.500 1 0 0.36 180  0 0 0  30  1 0 0 0 0
   4  7 -1  1 12  0  2  0  0  0  2  0.000  0.000 1.500 2 0 0.36 180  0 0 0  23  1 0 0 0 0
   5  7 -1  4 13  0  2  0  0  0  1 19.000 17.000 1.500 1 0 0.36   0  0 0 0  23  4 0 0 0 0
   6  6 -1  4  8  0  0  0  0  0  1 21.000 17.000 1.500 1 0 0      0 -1 0 0  23  4 0 0 0 0
       . . .
  30 38 -1  3  0  1  0  0  0  0  2 17.000  5.000 1.500 1 0 0      0 -1 0 0   0  6 0 0 0 0
  31 64  2 -1  0  0  0  0  0  0  2  0.000  0.000 1.500 1 0 0      0 -1 0 0   0  6 0 0 0 0
-999
```

❑ Section 17: Duct Junctions

Junction data are read by the jct_read( ) function and saved by the jct_save( ) function. The data are stored in the JCT_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The junction section starts with:

```
_njct      // number of junctions (I2)
```

This is followed by a data header comment line and then data for all _njct junctions.

For each junction the data line includes:

```
   nr        // zone number (I2); in order from 1 to _nzone
   flags     // zone flags – bits defined in contam.h (U2)
   pzn       // surrounding zone index (I2); converted to pointer
   pw        // wind coefficients index (I2); converted to pointer
   dnr       // duct segment number - use during read (I2)
   pk        // kinetic reaction index (I2); converted to pointer
   ps        // schedule index (I2); converted to pointer
   pc        // control node index (I2); converted to pointer
   pld       // level index (I2); converted to pointer
   X         // X-coordinate of ambient terminal [m] (R4) {Contam 2.1}
   Y         // Y-coordinate of ambient terminal [m] (R4) {Contam 2.1}
   relHt     // height relative to current level [m] (R4)
   T0        // initial junction temperature [K] (R4)
   P0        // initial junction pressure [Pa] (R4)
   wPset     // constant wind pressure [Pa] (pw==NULL) (R4)
   wPmod     // wind speed(?) modifier (pw!=NULL) (R4)
   wazm      // wall azimuth angle (pw!=NULL) (R4)
   u_Ht      // units of height (I2) {W}
   u_T       // units of temperature (I2) {W}
   u_dP      // units of pressure difference (I2) {W}
   icon      // icon used to represent flow path (U1) {W}
   ddir      // direction of terminal duct - to show wP (U1) {W}
   fdir      // positive flow direction - to show flow (U1) {W}
```

The junctions section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
19 ! duct junctions:
!J# f t z# w# dct k# s# c# l#    X     Y     relHt  T0 P0 wPset wPmod wazm u..
 1 3 3 -1  0   0  0  0  0  2  17.000 21.000 0.000 296.15 0 0 0 -1 0 0 2 0 162 2 2
 2 3 0 -1  0   0  0  0  0  2   0.000 0.000  0.000 296.15 0 0 0 -1 0 0 2 0 158 0 0
 . . .
19 3 1  1  0   0  0  0  0  2   0.000 0.000  0.000 296.15 0 0 0 -1 0 0 2 0 162 1 1
-999
```

❑ Section 18: Initial Junction Concentrations

Initial junction contaminant mass fractions are read by the zone_read( ) function and saved by the zone_save( ) function. The mass fractions are stored in the ZONE_DSC structures that are defined in contam.h for ContamW and they are stored in an array in ContamX.

The zone contaminants section starts with:

```
nn         // number of mass fraction that follow (I4)
```

*nn should equal _nzone * _nctm. This is followed by _nzone lines in order from zone 1 to zone _nzone.*

Each data line contains _nctm mass fractions (R4) in contaminant order.

The initial zone concentrations section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
38 ! initial concentrations:
! J#         C1         C2
   1 1.000e+000 2.000e+000
   2 1.000e+000 2.000e+000
   3 1.000e+000 2.000e+000
   4 1.000e+000 2.000e+000
   5 1.000e+000 2.000e+000
   6 1.000e+000 2.000e+000
   7 1.000e+000 2.000e+000
   8 1.000e+000 2.000e+000
   9 1.000e+000 2.000e+000
  10 1.000e+000 2.000e+000
  11 1.000e+000 2.000e+000
  12 1.000e+000 2.000e+000
  13 1.000e+000 2.000e+000
  14 1.000e+000 2.000e+000
  15 1.000e+000 2.000e+000
  16 1.000e+000 2.000e+000
  17 1.000e+000 2.000e+000
  18 1.000e+000 2.000e+000
  19 1.000e+000 2.000e+000
-999
```

❑ Section 19: Duct Segments

Duct data are read by the duct_read( ) function and saved by the duct_save( ) function. The data are stored in the DCT_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The path section starts with:

```
_ndct      // number of ducts (I2)
```

This is followed by a data header comment line and then data for all _ndct ducts.

For each duct the data line includes:

```
nr        // duct number (I2); in order from 1 to _ndct
flags     // duct flag values (I2)
pjn       // junction N index (I2); converted to pointer
pjm       // junction M index (I2); converted to pointer
pe        // duct flow element index (I2); converted to pointer
pf        // filter index (I2); converted to pointer
ps        // schedule index (I2); converted to pointer
pc        // control node index (I2); converted to pointer
dir       // positive flow direction on sketchpad (U1) {W}
length    // length of the duct segment [m] (R4)
Ain       // flow area at inlet end [m^2] - future (R4)
Aout      // flow area at outlet end [m^2] - future (R4)
Sllc      // sum of local loss coefficients (R4)
```

```
Lam       // laminar loss coefficient (R4)
u_L       // units for length (I2) {W}
u_A       // units for flow area (I2) {W}
```

The paths section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
18 ! duct segments:
! D#  f  n#  m#  e#  f#  s#  c# dir len Ain Aout sllc u..
   1  0   1   2   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   2  0   2   3   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   3  0   3   4   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   4  0   4   5   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   5  0   5   6   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   6  0   6   7   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   7  0   7   8   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   8  0   8   9   1   0   0   0   2 5 0.15708 0.15708 0 0 0
   9  0   9  10   1   0   0   0   2 5 0.15708 0.15708 0 0 0
  10  0  11   2   9   0   0   0   1 5 0.15708 0.15708 0 0 0
  11  0  12   3   8   0   0   0   1 5 0.15708 0.15708 0 0 0
  12  0  13   4   7   0   0   0   1 5 0.15708 0.15708 0 0 0
  13  0  14   5   6   0   0   0   1 5 0.15708 0.15708 0 0 0
  14  0  15   6   5   0   0   0   1 5 0.15708 0.15708 0 0 0
  15  0  16   7   4   0   0   0   1 5 0.15708 0.15708 0 0 0
  16  0  17   8   3   0   0   0   1 5 0.15708 0.15708 0 0 0
  17  0  18   9   2   0   0   0   1 5 0.15708 0.15708 0 0 0
  18  0  19  10   1   0   0   0   1 5 0.15708 0.15708 0 0 0
-999
```

❑ Section 20: Source/Sinks

Contaminant sources/sinks are read by the css_read( ) function and saved by the css_save( ) function. The data are stored in the CSS_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The source/sink section starts with:

```
_ncss     // number of source/sinks
```

This is followed by a data header comment line and then data for all _ncss source/sinks.

For each source/sink the data line includes:

```
   nr        // source/sink number (I2); in order from 1 to _ncss
   pz        // zone index (I2); converted to pointer
   pe        // source/sink element index (I2); converted to pointer
   ps        // schedule index (I2); converted to pointer
   pc        // control node index (I2); converted to pointer
   mult      // multiplier (R4)
   CC0       // initial mass fraction (some types) [kg/kg] (R4)
```

The source/sink section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
6 ! source/sinks:
! #  z#  e#  s#  c#  mult  CC0
  1   1   1   0   0     1    0
  2   1   2   0   0     1    0
  3   1   3   0   0     1    0
  4   1   4   0   0     1    0
  5   1   5   0   0     1    0
  6   1   6   1   0     1    0
-999
```

❑ Section 21: Occupancy Schedules

Occupancy schedules are read by the oschd_read( ) function and saved by the oschd_save( ) function. The data are stored in the OD_SCHD structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The occupancy schedule section starts with:

```
_nosch     // number of schedules (I2)
```

This is followed by a data header comment line and then data for all _nosch schedules.

For each schedule the first data line includes:

```
   nr        // schedule number (I2); in order from 1 to _nosch
   npts      // number of points (I2)
   name[]    // schedule name (I1) {W}
```

and the second line has:

```
   desc[]    // schedule description (I1) {W} may be blank
```

This is followed by npts lines of schedule data:

```
   time     // time-of-day [s] (hh:mm:ss converted to I4)
   pz       // zone index (I2); converted to pointer
```

The occupancy schedule section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
1 ! occupancy schedules:
1 3 OccDySched1
Occupant day schedule one.
 00:00:00 0
 00:12:00 4
 24:00:00 4
-999
```

❑ Section 22: Exposures

Exposure data are read by the pexp_read( ) function and saved by the pexp_save( ) function. The data are stored in the PEXP_DSC structures that are defined in contam.h for ContamW and simdat.h for ContamX.

The exposure section starts with:

```
_npexp     // number of exposures (I2)
```

This is followed by three lines of data for all _npexp exposures.

For each exposure the first data line includes:

```
nr        // exposure number (I2); in order from 1 to _npexp
gen       // = 1 if contaminants are generated (I2)
ncg       // number of contaminant generations (I2)
cgmlt     // contaminant generation multiplier [-] (R4)
inhmax    // peak inhalation rate [m^3/s] (R4) {unused}
inhsch    // weekly inhalation schedule index (I2)
bodywt    // body weight [kg] (R4) {unused}
u_inh     // units for inhalation rate (I2)
u_bw      // units for body weight (I2)
```

The second line has the exposure description:

```
desc[]    // exposure/person description (I1)
```

The third line has the indices of 12 occupancy schedules:

```
odsch[12] // vector of daily occupancy schedules – 12 indices
```

It is followed by ncg lines of contaminant generation data:

```
name      // species name (I1)
ps        // schedule index (I2); converted to pointer
cgmax     // peak generation rate [kg/s] (R4)
u_cg      // units of generation rate (I2) {W}
```

The exposure section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
2 ! exposures:
1 1 2 1 0.000334444 1 70 4 0
Occupant one.
 1 1 1 1 1 1 1 1 1 1 1 1 ! occ. schd
 C1 1 1 0 ! occ. gen
 C2 1 2 0 ! occ. gen
2 0 2 1 0.000668889 1 80 4 0
Occupant two.
 1 1 1 1 1 1 1 1 1 1 1 1 ! occ. schd
 C1 0 1 0 ! occ. gen
 C2 1 2 0 ! occ. gen
-999
```

❑ Section 23: Annotations

Annotations are read by the note_read( ) function and saved by the note_save( ) function. The data are stored in the NOTE_DSC structures that are defined in contam.h. Annotations are used only in ContamW; nothing is stored when this section is read in ContamX.

The annotations section starts with:

```
_nnote     // number of annotations (I2)
```

This is followed by data for all _nnote annotations:

For each annotation data line includes:

```
   nr        // annotation number (I2) {W}; in order from 1 to _nnode
   note[]    // annotation (I1) {W}
```

The annotations section is terminated with:

```
-999       // used to check for a read error in the above data
```

Example:

```
6 ! annotations:
1 PLR_type Airflow Elements
2 to force flows through PLR paths
3 QF_type Airflow Elements
4 DR_type Airflow Elements
5 PL_type Airflow Elements
6 FN_type Airflow Elements
-999
```

2.4.2 Weather File (.WTH)

Ambient weather conditions are made available to ContamX through the weather file which has a .WTH file extension. Use this file when performing transient simulations and you need to vary the ambient conditions with time. If you need to specify wind pressures that vary with time and spatially around the building envelope, you should consider using a WPC file (See Working with WPC Files).

The first line of the weather file is used to identify the type of file. It is exactly:

```
WeatherFile ContamW 2.0
```

The second line is a description of the file entered by the user:

```
desc[]         // file description (I1) {W}; may be blank
```

Succeeding lines contain:

```
StartDate      // first date for weather data (mm/dd → IX)
EndDate        // last date for weather data (mm/dd → IX)
```

These are followed by a header line for the date data:

```
!Date  DofW  Dtype  DST  Tgrnd
```

Day data for each date from StartDate to EndDate:

```
   date          // date (mm/dd → IX)
   dayofwk       // day of week [1=Sun … 7=Sat] (I2)
   daytype       // type of day for schedule reference [1-12] (I2)
   DST           // daylight savings time [0 / 1 = DST in effect] (I2)
   Tground       // ground temperature [K] (R4)
```

These are followed by a header line for the weather data:

```
!Date Time Ta  Pb  Ws  Wd  Hr  Ith  Idn  Ts  Rn  Sn
```

Weather data for each date from StartDate to EndDate:

```
   date          // date (mm/dd → IX)
   time          // time of day (hh:mm:ss → I4)
   tmpambt       // ambient temperature [K] (R4)
   barpres       // absolute barometric pressure [Pa] (R4)
   windspd       // wind speed [m/s] (R4)
   winddir       // wind direction [deg]: 0=N, 90=E, 180=S, ... (R4)
   humratio      // humidity ratio [g H20/kg dry-air] (R4)
   solhtot       // total solar flux on horizontal surface [W/m^2] (R4)
   solhdif       // diffuse solar flux on horizontal surface [W/m^2] (R4)
   tskyeff       // effective sky temperature [K] (R4)
   rain          // rain indicator [0 or 1] (I2)
   snow          // snow indicator [0 or 1] (I2)
```

The file description may not begin with a '**!**'. The StartDate and EndDate are used to verify that the file data covers the entire period to be simulated. The StartDate may not be later than the EndDate.

The weather data must start at time 00:00:00 on the StartDate and end at 24:00:00 on the EndDate. The times must be in consecutive order, but the difference between successive times need not be constant.

NOTE: Be sure to set the date formats as shown in the sample below when editing/saving tab-delimited files with a spreadsheet program

The ground temperature and solar flux through snow cover data will be used if the program is expanded for full thermal simulation. At present place holder values (e.g., 0) may be used.

Example:

```
WeatherFile ContamW 2.0
Weather for Jan1 - Jan3
1/1   1/3
!Date DofW  Dtype DST   Tgrnd
1/1   4     4     0     281.6
1/2   5     5     0     281.57
1/3   6     6     0     281.54
!Date Time     Ta      Pb      Ws   Wd   Hr    Ith  Idn  Ts       Rn Sn
1/1   00:00:00 275.15  100000  3.6  290  2.213 0    0    259.442  0  0
1/1   01:00:00 275.15  100000  3.1  300  2.213 0    0    259.442  0  0
1/1   02:00:00 275.15  100000  2.1  300  2.213 0    0    259.442  0  0
. . .
1/1   24:00:00 273.15  100000  4.6  320  1.915 0    0    256.627  0  0
1/2   01:00:00 273.15  100000  2.1  330  1.915 0    0    256.627  0  0
1/2   02:00:00 272.15  100000  4.1  330  1.763 0    0    255.899  0  0
. . .
1/2   24:00:00 272.15  100000  1.5  230  1.763 0    0    255.899  0  0
1/3   01:00:00 272.15  100000  1    210  1.763 0    0    255.899  0  0
1/3   02:00:00 272.15  100000  2.6  260  1.763 0    0    255.899  0  0
. . .
1/3   24:00:00 282.15  100000  5.7  230  3.607 0    0    268.353  0  0
```

2.4.3 Contaminant File (.CTM)

Transient ambient species data is made available to ContamX through the contaminant file which has a .CTM file extension. Use this file when performing transient simulations and you want the ambient concentration to be represented by a single value at each time step. If you need the ambient concentration to vary spatially as well over the building envelope, then you should use the .WPC file (see Working with WPC Files in Part 1).

The first line of the contaminant file is used to identify the type of file. It is exactly:

```
SpeciesFile ContamW 2.0
```

The second line is a description of the file entered by the user:

```
desc[]          // file description (I1) {W}; may be blank
```

Succeeding lines contain:

```
StartDate       // first date for contaminant data (mm/dd → IX)
EndDate         // last date for contaminant data (mm/dd → IX)
NumCont         // number of contaminants (I2)
I1 Name[1]      // name of contaminant 1 (I1 - 16 characters max)
I1 Name[2]      // name of contaminant 2 (I1)
...
Name[NumCont]   // name of contaminant NumCont (I1)
```

The last name is followed by a header line for the remaining data:

```
!Date  Time Name[1] Name[2] ... Name[NumCont]
```

Then comes concentration data for each time step:

```
   date           // date (mm/dd → IX)
   time           // time of day (hh:mm:ss → I4)
   conc[1]        // concentration of contaminant 1 (R4)
   conc[2]        // concentration of contaminant 2 (R4)
   ...
   conc[NumCont] // concentration of contaminant NumCont (R4)
```

The file description may not begin with a '!'. The StartDate and EndDate are used to verify that the file data covers the entire period to be simulated. The StartDate may not be later than the EndDate. The contaminant names are matched against the names of the contaminants to be simulated. Concentrations for names that do not match will be ignored.

The concentration data must start at time 00:00:00 on the StartDate and end at 24:00:00 on the EndDate. The times must be in consecutive order, but the difference between successive times need not be constant. Concentrations are given in units of mass (or density) of contaminant per mass (or density) of air, air being the sum of all species including non-contaminants.

NOTE: Be sure to set the date formats as shown in the sample below when editing/saving tab-delimited files with a spreadsheet program

Example:

```
SpeciesFile ContamW 2.0
Demo ctm file for Jan 1 - Jan 3
1/1     1/3      2
CO      CO2
!Date   Time     CO         CO2
1/1     0:00:00  9.17E-07   5.23E-04
1/1     1:00:00  9.17E-07   5.23E-04
```

```
1/1     2:00:00  9.17E-07   5.23E-04
. . .
1/2     24:00:00 9.17E-07   5.23E-04
1/2     1:00:00  9.17E-07   5.23E-04
1/2     2:00:00  9.17E-07   5.23E-04
. . .
1/3     24:00:00 9.17E-07   5.23E-04
1/3     1:00:00  9.17E-07   5.23E-04
1/3     2:00:00  9.17E-07   5.23E-04
. . .
1/3     24:00:00 1.41E-06   5.23E-04
```

2.4.4 Restart File (.RST)

Restart is an alternative to normal initialization. ContamX will either read the restart file or create it. Creation may occur only for transient simulations starting at 00:00:00 and ending at 24:00:00, on the same or different days. A restart file will have data at 24:00:00 of all days simulated. If the run is not cyclic, it will also have data at 00:00:00 of the first day simulated.

All code is contained in file restart.c with file 'globals'.

```
IX set_urst( IX flag );
   flag = 0:  initialize parameters.
   flag = 1:  open restart file _urst; write header data.
   flag = 2:  close restart file.
   flag = 3:  remove restart file.

void resout( IX date, I4 time );
   if _urst open, write current date/time restart data.

void resget( IX rstdate, I4 rsttime );
   read rstdate/rsttime restart data.

HEADER DATA:
    m[0]  = 0L;   /* TURBO C++ messes up the first bytes written; */
    m[1]  = (I4)_nzone;       /* therefore, send 4 unused bytes. */
    m[2]  = (I4)_npath;
    m[3]  = (I4)_nctm;
    m[4]  = (I4)_njct;
    m[5]  = (I4)_ndct;
    m[6]  = (I4)_ncss;
    m[7]  = (I4)_nctrl;
    m[8]  = (I4)_rcdat.date_0;
    m[9]  = (I4)_rcdat.date_1;
    m[10] = (I4)_rSizeData;
    m[11] = (not set)

m[1] thru m[7] allow ContamW to check for some changes in the project.
m[8] and m[9] are the date limits displayed to the user.
m[11] will allow reading all dates on the file -- could be used to
  create a selection box of available dates.

RESTART DATA:
For all AF_NODEs:
   R8 T - temperature
```

```
  R8 P - pressure
  R8 M - mass
  R4 Mf[_nctm] - mass fractions
For all AF_PATHs:
  R8 Flow[0] - primary flow
  R8 Flow[1] - secondary flow
  R8 dP - pressure drop
For CSS_DSC:
  CSE_EDS (I4)pss->local (stored as R4, converted to I4 in simulation)
  CSE_BLS (R4)pss->local
For CT_NODEs:
  SNSDAT: R4 oldsig
  PICDAT: R4 oldsig, R4 olderr
  HYSDAT: R4 oldsig
```

2.4.5 Continuous Values File (.CVF)

Control node values that change linearly in time are made available to ContamX through the continuous values file which has a .CVF file extension.

The first line of the CVF is used to identify the type of file. It is exactly:

```
ContinuousValuesFile ContamW 2.1
```

The second line is a description of the file entered by the user:

```
desc[]          // file description (I1) {W}; may be blank
```

The next two lines define the period covered by the file:

```
StartDate       // first date for DEF data (mm/dd → IX)
EndDate         // last date for DEF data (mm/dd → IX)
```

The next section specifies the control node names:

```
_nbvfn          // number of CVF node names
```

followed by _nbvfn lines consisting of node names:

```
name[]         // node name (I1)
```

The remainder of the file consists of data for all nodes for each date and time from StartDate to EndDate:

```
date         // date (mm/dd → IX)
time         // time of day (hh:mm:ss → I4)
value[1]     // value [?] for name[1] (R4)
...
value[_nbvf] // value [?]for name[_nbvf] (R4)
```

The node data must start at time 00:00:00 on the StartDate and end at 24:00:00 on the EndDate. The times must be in consecutive order, but the difference between successive times need not be constant.

The file description may not begin with a '`!`'. The StartDate and EndDate are used to verify that the file data covers the entire period to be simulated. The StartDate may not be later than the EndDate. Data elements on a single line are separated by tabs. The data must be in time-sequential order. The file values must be in the units needed for the signal created by the control node. Be sure to include a data line at the end of each day – 24:00:00.

Node names may not include imbedded blanks. Data for nodes that are not in the project file will be ignored. If a CVF node name in the project file is not included in the CVF, a fatal error will result. ContamW will assist the user by checking node names. ContamX will perform the name check before simulation begins.

Example:

```
ContinuousValuesFile ContamW 2.1
Sample CVF file
1/1 1/3
4
node1
node2
node3
node4
1/1 00:00:00 0.0 0.5 0.5 1.0
1/1 04:00:00 1.0 0.0 1.0 0.0
1/1 08:00:00 0.5 0.5 0.5 0.5
1/1 12:00:00 0.0 1.0 0.0 1.0
1/1 16:00:00 1.0 0.5 0.5 0.0
1/1 24:00:00 0.0 0.5 0.5 1.0
1/2 04:00:00 1.0 0.0 1.0 0.0
1/2 08:00:00 0.5 0.5 0.5 0.5
1/2 12:00:00 0.0 1.0 0.0 1.0
1/2 16:00:00 1.0 0.5 0.5 0.0
1/2 24:00:00 0.0 0.5 0.5 1.0
1/3 04:00:00 1.0 0.0 1.0 0.0
1/3 08:00:00 0.5 0.5 0.5 0.5
1/3 12:00:00 0.0 1.0 0.0 1.0
1/3 16:00:00 1.0 0.5 0.5 0.0
1/3 24:00:00 0.0 0.5 0.5 1.0
```

2.4.6 Discrete Values File (.DVF)

Control nodes values that change discretely in time are made available to ContamX through the discrete values file which has a .DVF file name extension.

The first line of the DVF is used to identify the type of file. It is exactly:

```
DiscreteValuesFile ContamW 2.1
```

The second line is a description of the file entered by the user:

```
desc[]          // file description (I1) {W}; may be blank
```

The next two lines define the period covered by the file:

```
StartDate       // first date for DEF data (mm/dd → IX)
EndDate         // last date for DEF data (mm/dd → IX)
```

The next section specifies the control node names:

```
_ndefn          // number of DVF node names
```

followed by _ndefn lines consisting of:

```
name[]          // node name (I1)
value           // initial value [?] (R4)
```

Succeeding lines present data whenever a node value changes:

```
date            // date (mm/dd → IX)
```

```
time            // time of day (hh:mm:ss → I4)
name[]          // node name (I1)
value           // new value [?] (R4)
```

The file description may not begin with a '!'. The StartDate and EndDate are used to verify that the file data covers the entire period to be simulated. The StartDate may not be later than the EndDate. Data elements on a single line are separated by tabs. The data must be in time-sequential order. More than one node may change at the same time with each node listed on a separate line. The file values must be in the units needed for the signal created by the control node. Each day should end with a line of the form: date 24:00:00 0 0.

Node names may not include imbedded blanks. Data for nodes that are not in the project file will be ignored. If a DVF node name in the project file is not included in the DVF, a fatal error will result. ContamW will assist the user by checking node names. ContamX will perform the name check before simulation begins.

Example:

```
DiscreteValuesFile ContamW 2.1
Sample DVF file
01/1 01/03
4
node1 0.0
node2 0.5
node3 0.5
node4 1.0
1/1 02:00:00 1 0.75
1/1 02:30:00 3 0.25
1/1 03:00:00 4 0.00
1/1 08:00:00 2 1.00
1/1 12:45:00 1 1.00
1/1 21:30:00 1 0.75
1/1 24:00:00 0 0.00
1/2 01:30:00 1 0.00
1/2 07:59:59 3 0.25
1/2 23:30:00 1 0.75
1/2 24:00:00 0 0.00
1/3 18:00:00 3 1.00
1/3 24:00:00 0 0.00
```

2.4.7 Simulation Results File (.SIM)

The results of the ContamX simulation are communicated back to the ContamW program in the simulation results file. This file will be created in the same directory as the project file with the name of the project file and the .SIM extension appended. The format of this file has not changed since the Contam96 DOS version of the program. It is a binary file for faster access and smaller size than a text file. It can be read and reported as text by the SIMREAD.EXE program available from NIST.

The first 16 lines of the simulation results file contain data (32-bit integers) to help assure that the results apply to the project file currently in ContamW and to set the array sizes necessary to process the results.

```
0           // spacer (to avoid a TurboC bug) (I4)
_nzone      // number of airflow zones (excluding ambient) (I4)
```

```
_npath      // number of airflow paths (I4)
_nctm       // number of contaminants (I4)
_time_list  // listing time steps [s](I4)
_date_0     // start of simulation - day of year (I4)
_time_0     // start of simulation - time of day (I4)
_date_1     // end of simulation - day of year (I4)
_time_1     // end of simulation - time of day (I4)
_pfsave     // if true, write path flow results (I4)
_zfsave     // if true, write zone flow results (I4)
_zcsave     // if true, write zone contaminant results (I4)
_nafnd      // number of airflow nodes (zones + junctions) (I4)
_nccnd      // number of contaminant nodes (zones + junctions) (I4)
_nafpt      // number of airflow paths (paths + ducts) (I4)
```

This is followed by _nafnd lines of airflow node cross-reference data:

```
   typ        // source of node [zone or junction] (U2)
   nr         // zone or junction number (U2)
```

The next _nafnd lines give the contaminant node cross-reference data:

```
   typ        // source of node [zone or junction] (U2)
   nr         // zone or junction number (U2)
```

The next _nafpt lines give the airflow path cross-reference data:

```
   typ        // source of path [path, duct, or leak] (U2)
   nr         // path, duct, or leak number (U2)
```

The simulation results for each day consist of:

The results for each time step consist of:

A line of time and ambient data:

```
dayofy    // day of year [1 to 365] (I2)
daytyp    // type of day [1 to 12] (I2)
sim_time  // time value [s] [0 to 86400] (I4)
Tambt     // ambient temperature [k] (R4)
P         // barometric pressure [Pa] (R4)
Ws        // wind speed [m/s] (R4)
Wd        // wind angle [deg] (R4)
CC[0]     // ambient mass fraction of species 0 [kg/kg] (R4)
...
CC[n]     // ambient mass fraction of species n [kg/kg] (R4)
```

A line of data for each airflow path:

```
nr        // path number; use as check (I2)
dP        // pressure drop across path [Pa] (R4)
Flow0     // primary flow value [kg/s] (R4)
Flow1     // alternate flow value [kg/s] (R4)
```

A line of data for each airflow node (excluding ambient):

```
nr        // node number; use as check (I2)
            T         // node temperature [K] (R4)
P         // node reference pressure [Pa] (R4)
D         // node air density [kg/m^3] (R4)
```

A line of data for each contaminant node (excluding ambient):

```
nr        // node number; use as check (I2)
CC[0]     // mass fraction of species 0 [kg/kg] (R4)
```

```
...
CC[n]     // mass fraction of species n [kg/kg] (R4)
```

The time step data is followed by summary data for the day.

It begins with the following line of ambient data:

```
dayofy    // day of year [1 to 365] (I2)
daytyp    // type of day [1 to 12] (I2)
Tamax     // maximum ambient temperature [k] (R4)
Tamin     // minimum ambient temperature [k] (R4)
Pavg      // average barometric pressure [Pa] (R4)
Wsmax     // maximum wind speed [m/s] (R4)
Wsavg     // average wind speed [m/s] (R4)
CC[0]     // maximum ambient mass fraction of species 0 [kg/kg] (R4)
...
CC[n]     // maximum ambient mass fraction of species n [kg/kg] (R4)
```

A line of data for each airflow path:

```
nr        // path number; use as check (I2)
dPmax     // maximum pressure drop across path [Pa] (R4)
Flowmax   // maximum primary flow value [kg/s] (R4)
0.0       // place holder (R4)
```

A line of data for each airflow node (excluding ambient):

```
nr        // node number; use as check (I2)
T         // node temperature [K] (R4)
P         // node reference pressure [Pa] (R4)
D         // node air density [kg/m^3] (R4)
```

A line of data for each contaminant node (excluding ambient):

```
nr        // node number; use as check (I2)
CCmax[0]  // maximum mass fraction of species 0 [kg/kg] (R4)
...
CCmax[n]  // maximum mass fraction of species n [kg/kg]
```

Note: this file requires that the structures in ContamW and ContamX be compiled using no greater than 2-byte member alignment (under Visual C++). The file is unreadable if the default structure member alignment is used.

2.4.8 Controls Log File (.LOG)

The output from *report control elements*, are written to the controls log file. This file will be created in the same directory as the project file with the name of the project file and the .LOG extension appended. For example if the project is *MyProj.PRJ*, ContamX will create the file *MyProj.LOG*. Values will be reported for each *listing time step* of a transient simulation. The file is tab-delimited so it can be easily imported into a spreadsheet program.

The first line of the Controls LOG file is a comment line that contains a list of the headers for each report control element. These headers are created by the user.

The second line is a comment line that contains column headings indicating Date, Time and the units of each reported value.

Succeeding lines contain:

Data from 00:00:00 of StartDate to 24:00:00 of EndDate:

At each time:

```
date          // date (mm/dd → IX)
time          // time of day (hh:mm:ss → I4)
output[1]     // outputs for each report control (R4)
output[2]
. . .
output[number of report controls]
```

2.4.9 Wind Pressure and Contaminant File (.WPC)

This file provides ambient pressure and contaminant concentrations values for every flow path that connects to ambient. See section Working with WPC Files in PART 1 for information on using this file in CONTAM.

The first line of the WPC file is used to identify the type of file. It is exactly:

```
WPCFile ContamW 2.1
```

The second line is a description of the file entered by the user:

```
desc[]         // file description (I1) {W}; may be blank
```

Succeeding lines contain:

```
NumPath        // number of flow paths (I2)
NumCont        // number of contaminants (species) (I2) [0 possible]
UsePres        // 1 = use pressures, 0 = don't (I2)
dtmin          // minimum time step (I2)
StartDate      // first date for WPC data (mm/dd → IX)
EndDate        // last date for WPC data (mm/dd → IX)
I1 Name[1]     // name of contaminant 1 (I1 - 16 characters max)
I1 Name[2]     // name of contaminant 2 (I1)
...
Name[NumCont]  // name of contaminant NumCont (I1)
```

The next `NumPath` lines describe each flow path:

```
number         // sequence number (I2)
X              // X-coordinate (R4)
Y              // X-coordinate (R4)
Z              // Z-coordinate (R4)
map            // mapping [0 = OK; 1 = error] (I2)
```

Data from 00:00:00 of StartDate to 24:00:00 of EndDate:

At each time:

First line:

```
date          // date (mm/dd → IX)
time          // time of day (hh:mm:ss → I4)
pres          // ambient pressure [Pa] (R4) (if pressures are used)
dens          // air density [kg/m3] (R4) (if pressures are used)
```

Second line: (if pressures are used)

```
pres[1]       // absolute ambient pressure at path[1] [Pa] (R4)
...
pres[NumPath] // absolute ambient pressure at path[NumPath] [Pa] (R4)
```

Third line (for first contaminant, if `NumCont > 1`):

```
conc[1]       // concentration at path[1] [kg/kg air] (R4)
```

```
...
conc[NumPath] // concentration at path[NumPath] [kg/kg air] (R4)
```

Fourth line (for second contaminant, if needed):

```
...
```

Last line (for contaminant `NumCont`, if needed):

```
conc[1]       // concentration at path[1] [kg/kg air] (R4)
...
conc[NumPath] // concentration at path[NumPath] [kg/kg air] (R4)
```

The file description may not begin with a '`!`'. The StartDate and EndDate are used to verify that the file data covers the entire period to be simulated. The StartDate may not be later than the EndDate.

The data must start at time 00:00:00 on the StartDate and end at 24:00:00 on the EndDate. The times must be in consecutive order, but the difference between successive times need not be constant.

Example:

```
WPCfile ContamW 2.1
For WPCtest3.prj
2          ! flowpaths
1          ! contaminants
1          ! use pressure flag
0          ! time step
01/01      ! start date
01/01      ! end date
C1
!nr        X          Y          Z          map
1          0.000      4.000      1.500      0
2          8.000      4.000      1.500      0
01/01      00:00:00   101325     1.204
101308.30  101306.30
0.0        1.0e-6
01/01      24:00:00 101325 1.2041
101308.30  101306.30
0.0        1.5e-6
```

2.4.10 Path Location Data File (.PLD)

The PLD (Path Location Data) File will consist of the locations of any flow paths that connect the building to the external environment, as well as additional information for directing the behavior of the EWC File Converter. This file is created by ContamW and stored in the same directory as the project file having the same name as the project file but with the .pld extension replacing the .prj extension. It is created as needed for use in comparing WPC file information with that in the CONTAM project file. (see Working with WPC Files in Part 1 of this document).

- The file is in an ASCII line-delimited format.
- The reference location, defined by (Xref,Yref,Zref), represents the EWC file location that is equivalent to the origin of the CONTAM coordinate system for the path locations. The user must coordinate between the EWC file and CONTAM file to determine this location's coordinate values.

- The rotation angle is about the Z (vertical) axis. The user must coordinate the direction of rotation between CONTAM and EWC file coordinates It is assumed there are no rotations about the X or Y axes.
- The six coordinates on line 6 define a bounding box that surrounds the flow paths within this file, which could be used to assist the EWC File Converter in rapidly excluding unneeded points (e.g., that are too distant from the building) during the mapping process. Note that this bounding box does not necessarily surround the entire building.
- The simulation time step provides a recommendation to the EWC File Converter for reducing the size of the WPC file, so that only the time steps that coincide with the simulation time step could be included (which could reduce interpolation required).
- Each flow path is uniquely identified by a combination of flow path type and flow path ID values.
- Comments begin with an exclamation point ('!') and may begin at the start of any line (so that the entire line will be ignored) or after all fields of a line (so that the remainder of the line will be ignored).
- For user readability, fields should be commented whenever possible.
- The Precision column for the real data types may be interpreted similar to a C scanf() statement's conversion specification.

HEADER SECTION:

Line	Field	Data Type	Precision	Notes
1	EWC filename	character		filename includes full path
2	WPC filename	character		filename includes full path
3	file comments	character		user-defined notes about this file
4	coordinate headings	character		this is one line of comments for the next section; set to "!Xref Yref Zref angle"
5	Xref	real	%7.3f	units of meters
5	Yref	real	%7.3f	units of meters
5	Zref	real	%7.3f	units of meters
5	rotation angle	real	%8.3f	rotation about Z axis in degrees; positive for CCW direction
6	latitude	real	%8.4f	units of degrees; sign is positive for north
6	longitude	real	%8.4f	units of degrees; sign is positive for east
7	bounding box headings	character		this is one line of comments for the next section; set to "!Xmin Xmax Ymin Ymax Zmin Zmax"
8	Xmin	real	%7.3f	units of meters
8	Xmax	real	%7.3f	units of meters
8	Ymin	real	%7.3f	units of meters
8	Ymax	real	%7.3f	units of meters
8	Zmin	real	%7.3f	units of meters
8	Zmax	real	%7.3f	units of meters

9	time step headings	character		this is one line of comments for the next section; set to "!step shift start end"
10	time step	integer		units of seconds
10	data shift	time		format of hh:mm:ss; this is the starting time for the EWC time steps when converted to WPC
10	start date	date		format of mm/dd
10	end date	date		format of mm/dd
11	wind pressures	integer		1: include pressures in WPC; 0: don't
11	number of species	integer		
11	species map tolerance	real	%7.3f	for use by EWC file converter file to match species between EWC and PLD files by molecular weight.

CONTAMINANT DEFINITION SECTION:

12	species headings	character		this is one line of comments for the next section; set to "!name m.wt"

The next (number of species) lines contain this data:

for each species…

	name	character	
	molecular weight	real	%6.2f

end for each species

FLOWPATH DEFINITION SECTION:

-	number of flow paths	integer		
-	path map tolerance	real	%7.3f	for use by EWC file converter to match flow paths that are more than this distance away; units of meters; this field is on the same line as the above field
	flow path headings	character		this is one line of comments for the next section; set to "!type ID X Y Z"

The next (number of flow paths) lines contain this data:

for each flow path…

	flow path ID number	integer		
	x	real	%7.3f	units of meters
	y	real	%7.3f	units of meters
	z	real	%7.3f	units of meters

end for each flow path

last line marker -999		integer		

Example:

```
C:\Program Files\Contamw2\Prjs\WPCcube.ewc ! EWC file
C:\Program Files\Contamw2\Prjs\WPCcube1.wpc ! WPC file
WPC description
```

```
! Xref    Xref    Yref    angle
  0.000   0.000   0.000   0.00
  0.0000  0.0000  ! no latitude/longitude
! Xmin    Xmax    Ymin    Ymax     Zmin    Zmax
  0.000   5.000   2.500   10.000   1.000   1.000
! step  shift     start   end
  0     00:00:00  1/1     1/1
  1     2         0.01 ! pressures flag, # species, mapping tolerence
! name  m.wt
  CO    28.00
  CO2   44.00
  4     1.00 ! number of flow paths and mapping tolerance
! id#   X         Y       Z
  1     5.000     10.000  1.000
  2     0.000     7.500   1.000
  3     0.000     5.000   1.000
  4     0.000     2.500   1.000
-999
```

2.4.11 ContamX Log File (CONTAMX2.LOG)

ContamX produces a "log file" summarizing its execution every time it is run. The log file was created primarily as an aid to the program developers, but it can also help the user to more effectively run the program. In particular, it can be used to optimize the performance of a particular simulation, and it records error messages to help the user or the program developers track down problems.

The log file is named CONTAMX2.LOG and is written into the directory containing CONTAMX2.EXE. It is therefore overwritten every time ContamX is run. This prevents an accumulation of such files but requires the user to review the file before a subsequent run replaces it.

The following example output is from a very large project (1.4 Mbytes) involving a steady-state calculation with 489 zones and 4246 duct junctions.

```
Program:  C:\ContamW2\ContamX2.exe : Thu Jan 02 10:33:02 2003
Project:  C:\ContamW2\PrjFiles\ManyDucts.prj : Thu Jan 02 10:16:04 2003
Time:  Thu Jan 02 13:19:28 2003
```

The first three lines of the log file report the full path\name of the program and the time it was created, the full path\name of the project file and the time it was created, and the time the program started running.

```
Reading project file: C:\Program Files\ContamX2\test.prj
Read PRJ : 3391868 bytes allocated, 1082748 freed, 2309120 net
Number of --
      contaminants:    0
     day schedules:    0
    week schedules:    0
           filters:    0
         reactions:    0
     wind profiles:    0
   source elements:    0
     flow elements:  317
     airflow nodes: 4735
     airflow paths: 7690
```

```
      source/sinks:    0
```

The second section of the log file summarizes reading the project file. It includes the total heap memory allocated, heap used temporarily and freed, and the net heap remaining in use. It notes the number of components in the simulation (in this case for airflows only). This section can also include an echo of the project file (the echo parameter on the first line of the project file) to locate any error messages that occur while reading the project file.

```
Set up simulation at:  Thu Jan 02 13:19:29 2003
Airflow Equations:
4734 variable pressure nodes
   1 constant pressure nodes
  Number of equations: 4735
    Non-zero elements: 17427
Initial fill fraction: 0.001
 Matrix Profile Analysis:
      Average Profile: 656.94
            Bandwidth: 4229
Determining equation reordering:
Equations will be reordered.
      Average Profile: 204.62
            Bandwidth: 891
 Skyline Matrix:
       Number of rows: 4734
       Lower Triangle: 968667 elements
       Upper Triangle: 0 elements
        Fill fraction: 0.043
Solve nonlinear equations by Newton-Raphson with variable trust region.
Solve Jacobian simultaneous linear equations by symmetric skyline method.
Arrays: 12080012 bytes allocated, 1648272 freed, 10431740 net
```

The third section of the log file summarizes the creation of the arrays for solving the airflow and mass fraction equations. In this case the skyline method with equation reordering (the default) is being used to solve the 4734 simultaneous non-linear airflow equations. Equation reordering reduces the average profile by a factor of three resulting in a symmetric matrix containing 968,667 elements. This is most of the additional heap allocation.

```
Begin simulation at:  Thu Jan 02 13:19:29 2003
Start: 12110772 bytes allocated, 1679032 freed, 10431740 net
Running steady-state initialization:
 DATE: Jan01   time: 00:00:00
```

The fourth section echoes varying amounts of information during the simulation. It can echo the weather, contaminant, and other input files based on the echo parameter in the project file. It can also record the progress of a transient simulation and record the values being written to the simulation results file, based on the value entered for the "list" parameter which is toward the end of the run control section of the project file.

```
Time to perform simulation: 123.86 s
     0 time steps
     1 calls to SolveAf()
    31 calls to FillAf()
    22 calls to luf_sky_s()
    22 calls to lus_sky_s()
```

```
  2676 calls to cubic(N)

facBins: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
rangeDWC: 10062 lam, 2459 trns, 141529 turbulent

End: 12110772 bytes allocated, 1679032 freed, 10431740 net
Final: 12110772 bytes allocated, 12110772 freed, 0 net
Time:  Thu Jan 02 13:21:33 2003
```

The final section summarizes the calculations, the simulation time, and heap usage. The simulation time is accurate to 1/20th second, but may include other programs operating under MS Windows. There is one call to solve the airflows which required 22 iterations solving the simultaneous linear equations. The program frees all allocated heap memory as indicated in the next to last line where the number of bytes freed is identical to that allocated. The last line is the time at which the program terminated. Subtracting the time when the program started gives 125 seconds total run time. All but about one second was spent solving the airflow equations.

2.4.11.1 Error Messages

The log file records all error messages displayed on the screen. They are of the form:

```
severity  ( file, line )
message
```

where

`severity` is a word describing the importance of the error,

`file` is the name of the C source file where the error message originates,

`line` is the number of the line in that file, and

`message` describes the error to the user.

2.4.11.2 Evaluation of Alternate Solution Methods

ContamX provides alternate methods to solve the simultaneous non-linear equations and the simultaneous linear equations. These alternatives can be used when a problem is encountered with the default methods, and they may provide better performance. Here we will try several of these methods, on the same large project presented previously, as an exploration of their performance and the use of the log file to improve performance. First we will use the simple under relaxation method to solve the simultaneous non-linear equations. The simulation summary indicates:

```
Time to perform simulation: 107.10 s
     1 calls to SolveAf()
    20 calls to FillAf()
    19 calls to luf_sky_s()
    19 calls to lus_sky_s()
```

This is a small (13.5%) improvement because there are 3 (15%) fewer N-R iterations. The simple trust region is still recommended as being generally more reliable.

The default method for solving the simultaneous linear equations includes reordering the equations to reduce the profile of the Jacobian matrix. The following results show the impact of *not* reordering the equations.

```
Skyline Matrix:
       Number of rows: 4734
       Lower Triangle: 3109960 elements
       Upper Triangle: 0 elements
        Fill fraction: 0.139
Arrays: 28843652 bytes allocated, 1281568 freed, 27562084 net
...
Time to perform simulation: 1394.17 s
     1 calls to SolveAf()
    31 calls to FillAf()
    22 calls to luf_sky_s()
    22 calls to lus_sky_s()
Final: 28874412 bytes allocated, 28874412 freed, 0 net
```

The number of values stored in the Jacobian has more than tripled (970,000 to 3,100,000), allocated memory has increased by about 17,000,000 bytes, and – most importantly – the simulation time is 11.3 times longer. The gains will not be as dramatic for smaller problems and for those with well-banded Jacobians as can arise from tall buildings. However, the time for reordering is so small, that it should almost always be tried. The reordering algorithm is not totally reliable – sometimes it fails and a few crashes have been observed.

The alternate method for solving the simultaneous linear equations is a preconditioned conjugate-gradient (PCG) algorithm. This is an iterative method (the skyline method is direct) and may not converge. The following results were achieved for this case:

```
Arrays: 4470060 bytes allocated, 1281568 freed, 3188492 net
...
Time to perform simulation: 74.92 s
     1 calls to SolveAf()
    32 calls to FillAf()
    23 calls to sa_pcg()
 16666 sa_pcg() iterations
End: 4500820 bytes allocated, 1312328 freed, 3188492 net
```

There is a fairly significant (39.5%) improvement in solution time over the default case. There is a more dramatic savings (69%) in net allocated memory. The PCG method always uses less allocated memory than the skyline method, although that is usually not a problem – especially for relatively small projects. The PCG method is also usually slower than skyline for small projects. The PCG method should be tested when you have a large problem that will require several runs for a full analysis.

2.4.12 ContamW Configuration File (CONTAM.CFG)

Each time ContamW is run, it checks for the existence of the contam.cfg file. This file contains the default settings to use when the program is run (see Configuring CONTAMW in the 2.0 User Manual). All of the parameters that appear in the configuration file can be modified using the Options… selection of the View menu except for the EWC-to-WPC file converter. The value saved will be that which you have set in the WPC File… selection of the Weather menu. To reset the converter name back to "null," you must edit the configuration file manually.

The first line of the file is used to identify the type of file. It is exactly:

```
ConfigFile ContamW 2.1   ! file type identification
```

The next five lines are comments explaining font selection.

The seventh line is the list of font sizes to be made available for zooming the SketchPad

```
nFonts          // number of fonts available (1-7)
```

followed by nfont *values of font sizes (allowed values are 1, 2, 3, 4, 5, 8, 16)*

```
font(1) ... font(nfonts)
```

The next line indicates the default set of units to use (SI or IP)

```
DefUnits            // default units (0, 1) (I4)
```

The next line indicates the default flow units to use

```
defFlowUnit         // default flow units (0 - 8) (I4)
```

The next line indicates the default zone temperature and units

```
Tdef                // default zone temperature (R4)
Tunits              // temperature units (0 – 3) (I4)
```

The last line indicates the file path of an EWC-to-WPC file converter set to "null" if not used

```
null                // full pathname of file converter
```

Example:

```
ConfigFile ContamW 2.1   ! file type identification
! anything after an exclamation mark is treated as a comment.
! number of fonts, N, followed by N valid font sizes in increasing order.
! 7  1 2 3 4 5 8 16  ! all valid sizes; size 8 is required.
! 6  1 2 3 5 8 16 -or- 5  1 2 4 8 16  are good choices.
! if the size 1 font crashes the program, remove it.
  7  1 2 3 4 5 8 16  ! selected fonts.
  0     ! default units: 0 = SI, 1 = IP.
  4     ! default flow units: 0 = kg/s, 1 = scfm, 2 = sL/s, 3 = sm3/s,
        !   4 = sm3/h, 5 = lb/s, 6 = sft3/h, 7 = sL/min, 8 = kg/h.
 20 2   ! default zone temperatures in units:
        !   0 = K, 1 = R, 2 = C, 3 = F.
null    ! EWC to WPC converter
```

2.5 DATA STRUCTURES

Most of the data structures are defined in the files simdat.h and selmts.h. Global variables are defined in the sglob.h and sxtrn.h files. Such variables are indicated by a leading underscore in the variable name. Linked lists are used extensively to store the data describing the building airflow and contaminant components. Vectors and arrays are used in the solution of the linear and nonlinear equations.

PART 3 – CONTAMW PROGRAM DOCUMENTATION

3.1 INTRODUCTION

This part presents an overview of the program structure, data and logic of the ContamW, CONTAM's graphical user interface (GUI). It does not present the functions in as much detail as was done for the solver, because of the large number of files and functions associated with the GUI. The intent is to present enough information to provide the user of this document with a starting point to understanding the program structure. The GUI provides a means of creating and viewing the multi-zone model, and creates the PRJ file for use by CONTAM's simulation engine, ContamX. The GUI also serves as a means to view the simulation results that are output by ContamX.

3.2 DEVELOPMENT ENVIRONMENT

ContamW is an event-driven program developed solely for the Windows platform using the WIN32 Application Programming Interface (API) and written in the C programming language. Version 2.1 was developed using Microsoft Visual C++ versions 6.0 and .NET. As of the release of this document, the source code consists of 87 c-code files (.c), 33 header files (.h), 8 context-sensitive help header files (.hh), and 16 resource-related files. The graphical charting routines were developed using a third-party charting tool OlectraChart 6.0. This software is required in order to build the Visual Studio project.

3.3 PROGRAM STRUCTURE

The figure below shows the GUI which is made up of two windows – one "superimposed" on top of another. The *SketchPad* window is the large white portion that is "superimposed" on the larger *Main* window.

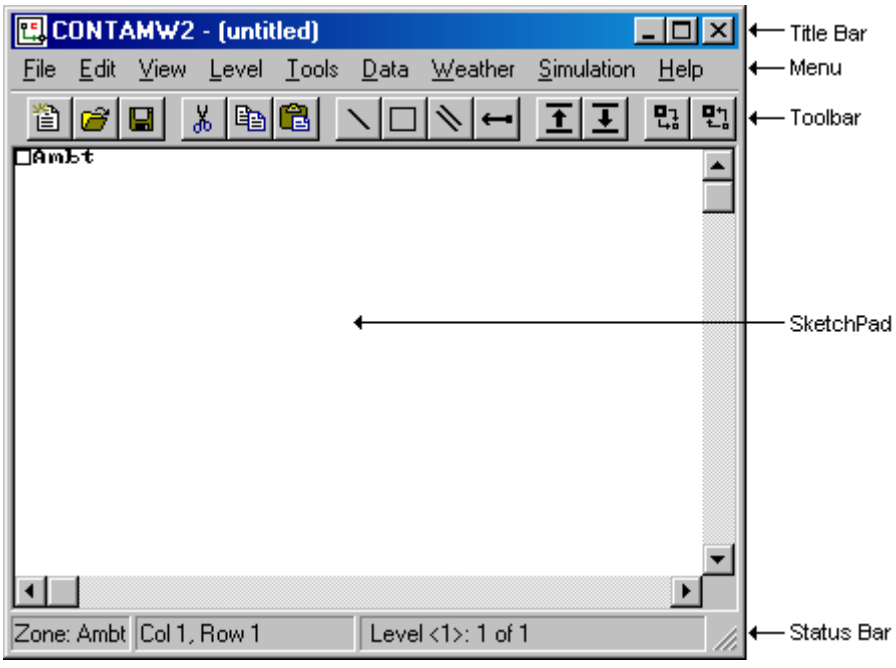


Figure – The CONTAMW GUI

3.3.1 Main Program and Message Loop

Every Windows program must have a *WinMain()* function which is the entry point of the program. This function is equivalent to the main() function in a standard C program. In ContamW, this function is located in the file *contam.c*. *WinMain()* performs program initialization and executes the main message loop of the program.

As this is an event-driven program, it contains a message loop that gets and dispatches messages from and to the operating system respectively. This loop gets messages that are directed to the ContamW program from the Windows message queue then dispatches the message back to the operating system which then calls the appropriate *window procedure* of ContamW for processing.

3.3.2 Window Procedures

Each window displayed by ContamW, including dialog boxes and property sheet pages, has an associated *window procedure*. Window procedures are called by Windows after ContamW dispatches messages associated with a particular window back to the operating system. These messages contain parameters that indicate to the operating system which window procedure to call, a message identifier, and message-specific parameters as required. Window procedures contain sections of code known as *message handlers* that are executed when the window procedure receives a particular message.

ContamW contains approximately 90 window procedures. Two of these procedures, *WndProc()* and *SketchPadWndProc()*, are associated with the two main windows of the program and the remaining procedures are associated with individual dialog boxes, property sheet pages and miscellaneous windows that display charts and simulation results. *WndProc()* handles the menu and associated toolbar commands and passes keyboard messages to the SketchPad window procedure *SketchPadWndProc()*. *SketchPadWndProc()* handles all SketchPad related commands including drawing walls, ducts and controls and placing icons onto the SketchPad.

3.4 PROGRAM DATA

Before describing the ContamW data structures, it is important to understand the basic philosophy of the program. ContamW is not meant to depict an exact physical representation of a building, but to portray a building in a manner that is representative of the multizone modeling perspective. Therefore, the physical dimensions of the representations of building zones are not important, but the interconnectivity or topology of the zones is important. The ContamW SketchPad provides a means of describing a building in a manner that constrains the representation to the multizone modeling domain.

3.4.1 SketchPad Data

From the users perspective, the SketchPad consists of a two-dimensional grid of equal-sized cells in which icons can be placed. The SketchPad is actually represented as a set of global 2D arrays having equal dimensions indicating the width (columns) and height (rows) of the SketchPad. These arrays are used to indicate the symbols located on the SketchPad for each building level and to maintain references to lists of data elements that contain the detailed information describing the building elements that make up each building level. The following is a list of the SketchPad arrays along with a description of the values stored in each array. Values are defined in *contam.h*. In the list below, the first array in each set contains information related to the

building level currently displayed on the SketchPad, and the second (preceded by _SL) contains information related to the level below the current level, i.e. the sublevel.

_**Sketch**[row][col] _**SLSketch**[row][col]	Contains identifiers indicating a symbol for *Zones*, *Flow path*, *Simple AHS*, *Supply*, *Return*, *Note*, *Source Sink and Exposure* (0 => empty cell)
_**Walls**[row][col] _**SLWalls**[row][col]	Contains identifiers indicating the wall symbol. Symbol values from *WL_EW* to *WL_NESW*.
_**Zones**[row][col] _**SLZones**[row][col]	Indicates zone number for every cell on the SketchPad. AMBT 32766 => ambient zone ZNDF -2 => undefined zone WALL 32767 => wall location
_**Paths**[row][col] _**SLPaths**[row][col]	Contains the id number of each symbol that represents a building component. The following types of components warrant values in the _Paths array: Airf*low paths*, *Ducts*, *Junctions*, *Zones* and *Notes.*
_**Ducts**[row][col] _**SLDucts**[row][col]	Contains identifiers indicating a symbol for all *Ducts*, *Junctions* and *Terminals*. Values are from *DCT_EW* to *IOJ_CB*.
_**Links**[row][col] _**SLLinks**[row][col]	Contains identifiers indicating controls-related symbols. Values are from *CL_EW* to *CTRL_P*.
_**Ctrls**[row][col] _**SLCtrls**[row][col]	Contains a ctrl/link number for each control symbol in the corresponding _Links array.

Table – Global SketchPad arrays

An example of a project file and associated SketchPad arrays is presented in spreadsheet form in Appendix 3A at the end of this section.

3.4.2 Building Organization

ContamW buildings (projects) are organized by building levels that contain building component information. This organization is represented by a doubly linked list of level data structures (*LEV_DATA*) as illustrated in the figure below. This list is used to populate the SketchPad arrays whenever the user selects a different level to display. A global pointer is used to access the current level and the levels above and below the current level are accessed via each level's pointer to the level above and below. Each level contains a pointer to a list of icon data for all of the icons contained on the level. Icon data structures (*ICON_DAT*) contain information that includes the component number, icon identifier, and column and row in which the icon is located on the SketchPad, i.e., in the SketchPad arrays.

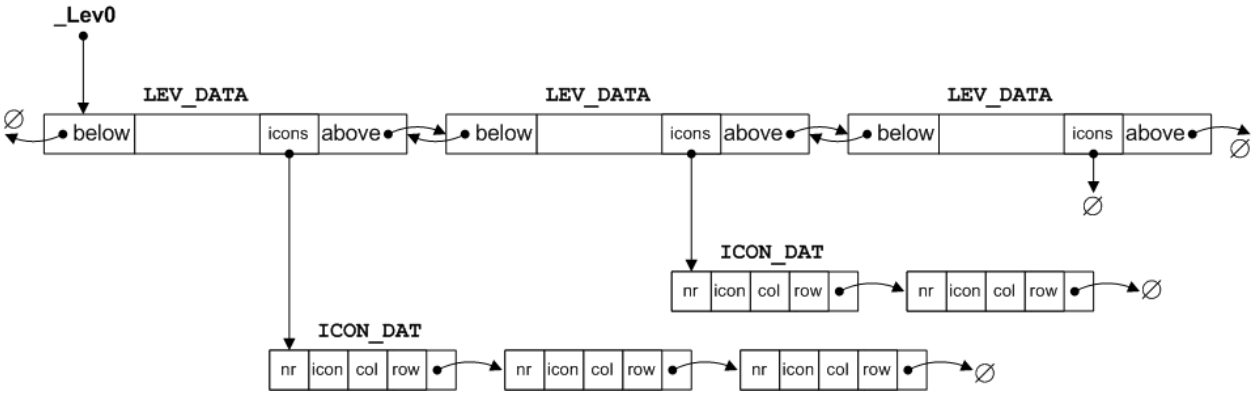


Figure – Schematic of Building Level and Icon Data Storage

3.4.3 Building Component and Element Data

The data used to describe a building in ContamW is basically divided into two types: building *component* and building *element* data. In terms of how the data is managed by ContamW – these data are referenced-by-number (RefByNum) and referenced-by-name (RefByName) respectively. RefByName data are referenced by a unique, user-defined name, and RefByNum data are referenced by a number assigned by ContamW.

RefByNum data are those that are considered specific to a building (project) and include *zones*; *airflow paths*; *simple air handling systems*; *duct segments*, *junctions* and *terminals*; *source-sinks*; *occupants* and *controls*. These components only exist in relation to a given building, i.e., they physically relate to other components of the building and are represented on the SketchPad by individual icons. RefByName data are those types of data that can be shared between building components and even between different projects via ContamW libraries. RefByName elements include *airflow elements*, *duct flow elements*, *wind pressure profiles*, *species*, *source-sink elements*, *filter elements*, *kinetic reaction elements*, *non-occupant schedules* and *annotations*.

RefByNum data can refer to RefByName data, and more than one RefByNum component can refer to the same RefByName element. For example an *airflow path* (e.g. a doorway) connects two zones of a building is defined in part by its location within the building. The airflow characteristics of this *airflow path* are represented by a pressure-flow model characterized by an *airflow element*. There may be several doors having the same airflow characteristics; therefore, each airflow path (e.g. door) can refer to the same airflow element.

Building component data are stored in linked-lists as shown in the figure below that illustrates the concept for airflow paths and elements. These lists are referred to by and accessed through

arrays of pointers (e.g. *_PathList*) that are used to maintain the components in order according to their location within the building. Building components are numbered starting at the top level in the upper left hand corner of the SketchPad moving left to right and down the SketchPad then proceeding down through each level in the same manner. Whenever a project file is saved, reordering of the array of pointers and renumbering of the building components will be performed as necessary.

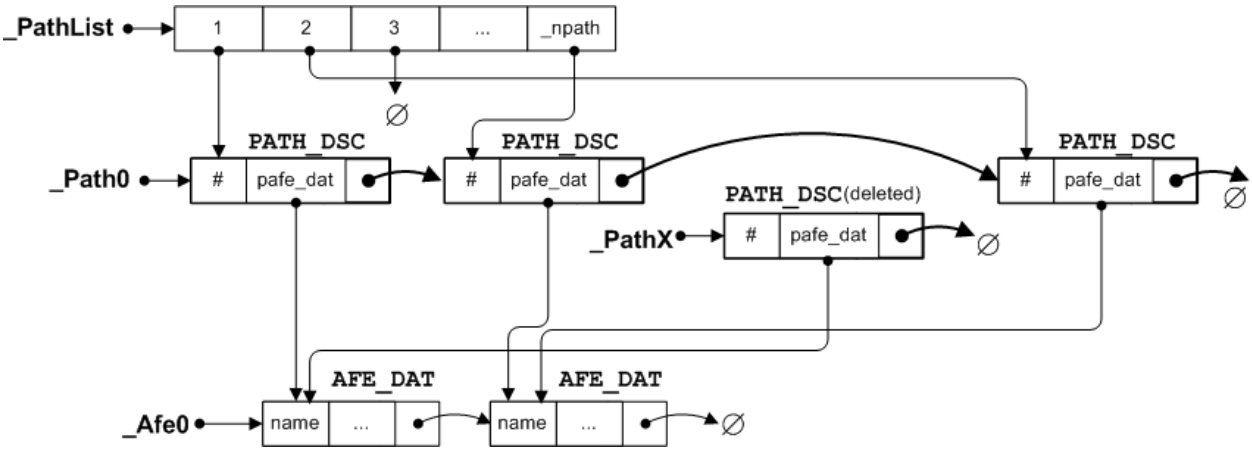


Figure – Schematic of Airflow Path Component (PATH_DSC) and Element (AFE_DAT) Data Storage

The figure above shows some of the global variables that are used to maintain the list of airflow paths (components) and airflow elements that are referenced by the paths. *_PathList* is the array of pointers used to maintain the SketchPad order of the paths, *_Paths0* is a pointer to the head of the linked list of *PATH_DSC* structures. *_PathX* is a pointer to the head of a list of paths structures that have been deleted, but could be reused when new paths are added.

The routines used to create new paths are contained in the file *paths.c*. These routines are typical of the building component management routines:

- Path creation functions: *path_new()*, *path_dflt()*, *path_add()*, *path_put()*
- Path deletion functions: *path_del()*, *path_delete()*
- Path editing functions: *path_old()*, *path_get()*

The routines for managing airflow element related data are found in *afedlg.c* (airflow element specific) and *cutils.c* (building elements in general).

- Airflow element creation functions: *afelmt_new()*, *elmt_insert()*
- Airflow element deletion functions: *elmt_delete()*
- Airflow element editing functions: *afelmt_old()*, *elmt_replace()*

These data are managed by a set of lower level routines that perform all memory allocation/deallocation for ContamW. These routines are located in the file *heap.c*. These memory management routines are used to minimize the number of actual calls to the standard-C *malloc()* and *free()* functions by maintaining memory used by ContamW in larger memory blocks with which the routines in *heap.c* operate.

3.5 PROGRAM LOGIC

As previously indicated, ContamW is an event-driven program, which basically means that the program functions that are executed depend on user-generated commands, i.e., user events. These events fall into the following basic categories: saving/retrieving project files, drawing building components, editing building components, running simulations and working with simulation results. The following sections describe the program logic that handles these events.

3.5.1 Message (Event) Handlers

Selecting an item from the menu or double-clicking on a SketchPad icon are examples of events for which message handlers are required. In ContamW, all menu selection events are handled by the *WM_COMMAND* message handler *WndProc_OnCommand()* of *WndProc()*. Each menu selection is associated with a resource identifier that is used as a case selector of a switch statement in the *WM_COMMAND* message handler. For example, if the user selects the File-Open command, the *WM_COMMAND* message is sent with the identifier of the File-Open menu resource *IDM_FILE_OPEN*. In the case of a double-click mouse event on the SketchPad window, both the *WM_LBUTTONDOWN* and *WM_LBUTTONDBLCLK* commands are sent to *SPWndProc_OnLButtonDown()* message handler of *SketchPadWndProc()*.

3.5.2 Saving and Retrieving Project Files

CONTAM project files are ASCII files that maintain all information related to a CONTAM project. The project file format is described in detail in the section Project File (.PRJ). Project file related commands are executed via the File menu and are handled within the *WM_COMMAND* message handler of the main window procedure *WndProc()*. The File menu identifiers are all of the form *IDM_FILE_X* where *X* is one of the following *NEW* , *OPEN*, *SAVE*, *SAVE_AS*, or *EXIT*. File input (reading) and writing (saving) are handled by the *prj_read()* and *prj_save()* functions contained in the files *prjread.c* and *prjsave.c*. These two functions call lower level functions each of which handles a specific section of the project file as described in the project file documentation.

There are a few other features of note related to project file processing. If the user attempts to open a project file of an older version, conversion functions will be called to convert to a format compatible with the current version of the program. These routines are located in the file *c10toc20.c*. If the user clicks the OK button after viewing a building component data dialog box, a global flag *_saveprjf* is set to indicate that the file should be saved and a warning will be displayed if the user attempts to exit the program with this flag set. Also, the project file will be saved automatically when the user attempts to perform a simulation, because the simulation program reads the project file and it is a separate executable from the GUI.

3.5.3 SketchPad Drawing

Icons are placed upon the SketchPad using either one of the four drawing tools or the popup icon placement menu.

3.5.3.1 Drawing Tools

There are four drawing tools: line and box for walls, ducts and control links. Tool selection is performed via either the Tool menu or associated toolbar buttons. The message handlers

contained in *WndProc()* associated with each tool set a global flag to indicate which tool is selected: *_bWall*, *_bBox*, *_bDuct* or *_bLink*.

The actual drawing process is activated once the user clicks the left mouse button (or hits the Enter key) at which point the drawing mode is activated and indicated by setting the flag *_bDraw*. When in the drawing mode, all mouse commands and keyboard arrow key commands are captured by the SketchPad window and converted to the ContamW-specific message *CT_DRAW* handled by the function *SPWndProc_OnCtDraw()*. *SPWndProc_OnCtDraw()* then calls specific drawing functions depending on the selected drawing tool as indicated by the drawing tool flags. Drawing is finalized by clicking the left mouse button activating the message handler *SPWndProc_OnLButtonDown()*. This message handler will then call the drawing tool dependant routines to validate the drawing and place the proper icons into SketchPad arrays. If the drawing was valid then the screen can be updated to reflect the placement of the building components, e.g., walls, ducts or control links. This is performed via the *InvalidateRect()* command that causes the SketchPad window to redraw or repaint itself. SketchPad "painting" is performed via the *WM_PAINT* message handler *SPWndProc_OnPaint()*.

The following is a pseudo-code outline of the drawing process, and the following list indicates the functions and their locations within the source code files.

PseudoCode for the Wall Drawing Process

Select Tool

```
WndProc_OnCommand ( IDM_TOOLS_DRAWWALL )  // User: selects drawing tool
{  _bWall = true;                         // tool selection flag
   SPWndProc_OnCtSwitchCursor( 1 );       // display drawing cursor
}
```

Begin Drawing

```
SPWndProc_OnLButtonDown( )                // User: clicks LMB
{  if( !_bDraw ) _bDraw = true;           // set drawing flag
}
```

Drawing

```
SPWndProc_OnKey( left | right | up | down | enter | esc )
SPWndProc_OnMouseMove( x, y )
{  SendMessage( SketchPad, CT_DRAW, dir, increment );
}
```

Finish Drawing

```
SPWndProc_OnLButtonDown( )                // User: clicks LMB
{  if( _bDraw )
      if( wall_check() == 0 )             // validate wall drawing
      {  walls_set(0);                    // set icons in _Walls[][]
         zones_set(0);                    // set icons in _Zones[][]
         InvalidateRect( SketchPad );     // send WM_PAINT message
      }
}
```

SketchPad Painting – Display of SketchPad arrays on the screen

```
SPWndProc_OnPaint( )          // SketchPad window WM_PAINT message handler
{
   sketch_redraw()
   {  for( for each cell of SketchPad )
      {  sk_draw_symbol()
         {  grblank();                    // set background color of icon
            sk_draw_icon()                // get icons from SketchPad arrays
            {  grputc(icon, color)
               {  SetTextColor(color);    // Windows API function call
                  TextOut(icon);          // Windows API function call
               }
            }
         }
      }
   }
}
```

TextOut()

This is a Windows Graphics Device Interface (GDI) function that outputs text to a given device context which in this case is the SketchPad window. The fonts for the device (*WALTON01-16.FON*) are initialized at program startup by the *InitFont()* function call from within S*PWndProc_OnCreate()*. The ContamW installation program installs these fonts into the system fonts folder.

Function	File Name
WndProc_OnCommand( ) IDM_TOOLS_DRAWWALL	WndProc.c
SPWndProc_OnCtSwitchCursor( ) SPWndProc_OnLButtonDown( ) SPWndProc_OnKey( ) SPWndProc_OnMouseMove( ) SPWndProc_OnCtDraw( ) SPWndProc_OnPaint( )	SPWndPro.c
walls_set( ) zones_set( ) sketch_redraw() sk_draw_symbol() sk_draw_icon()	sketch.c
wall_check( )	walldraw.c
grblank( ) grputc( ) InitFont( )	wutils.c
SetTextColor( ) TextOut( )	Windows API

Table – SketchPad Drawing Related Functions

3.5.3.2 Icon Placement

Some building components are drawn onto the SketchPad via the pop-up icon placement menu. This menu contains a list of building components that are represented by specific icons. The menu selections are enabled/disabled based on context-sensitive rules for icon placement.

The menu is activated when the user clicks the right mouse button (RMB) on the SketchPad. This causes the *WM_ONRBUTTONDOWN* message to be sent to the SketchPad window which is handled by the *SPWndProc_OnRButtonDown()* message handler. This in turn calls both *SetPopMenu()* and *TrackPopMenu()* functions. *SetPopMenu()* is a ContamW function that implements the context-sensitive rules of icon placement, and *TrackPopMenu()* is a Windows API function that displays the actual menu wherever the mouse pointer is when the RMB is clicked. When the user selects an item from the icon placement menu a *WM_COMMAND* message is sent to the SketchPad window along with a menu command identifier. The *SPWndProc_OnCommand()* message handler function contains a case for each icon placement menu selection of the form *IDM_POP1_X* where *X* stands for *ZONE*, *AMBIENT*, *PHANTOM*, *PATH*, *AHS*, *INLET*, *OUTLET*, *SS*, *EXPOSURE*, or *NOTE*. Each of these message handlers places the corresponding icon identifier in the appropriate SketchPad array then calls the *sk_draw_symbol()* function to display the new icon on the SketchPad window. The icon will be red in color to indicate that the icon is undefined, i.e., it is not yet associated with a building component. At this point a value indicating an undefined building component will also be placed into the appropriate SketchPad arrays if necessary, e.g., a value of ZNDF (-2) indicating an undefined zone will be placed in the *_Paths* array and in every cell of the *_Zones* array that is enclosed by the walls immediately surrounding the zone icon.

3.5.4 Creating and Editing Building Components

Once an icon is placed on the SketchPad a building component must be created and associated with it. This is done via dialog boxes and property sheets associated with the type of building component represented by the particular icon.

When the user double clicks on an undefined icon on the SketchPad the *WM_LBUTTONDOWN* message is sent to the SketchPad window with the *fDoubleClick* parameter set to true. The *SPWndProc_OnLButtonDown()* function checks the SketchPad arrays in a hierarchical fashion to determine the type of icon for which to display properties. Having determined the type of undefined icon, a temporary building component data structure is allocated with default values. This temporary structure is passed as a parameter to the associated component dialog box procedure (a type of window procedure) to be modified as required by the user. If the user selects "OK" on the dialog box, control returns to the *SPWndProc_OnLButtonDown()* message handler, and a permanent copy of the data structure representing the building component is made and added to the corresponding list of building components (see Building Component and Element Data). At this point a number value is assigned to the building component by ContamW and placed into the appropriate SketchPad array at the proper location.

Editing of existing building components is accomplished in much the same way as creating a new one. The user double clicks on the desired icon and instead of creating a new, default component, a copy is made of the existing component and sent to the appropriate dialog box procedure.

3.5.5 Running Simulations

ContamW is used to develop a set of equations that represents a building as a multizone airflow and contaminant transport network. The GUI provides the means to develop a schematic representation of the building in the level of detail required by the multizone modeling paradigm. The equation solver then takes this schematic representation and converts it into a set of airflow and contaminant transport equations.

The equation solver, *contamx2.exe*, is a stand-alone executable that operates directly on the project file created using the ContamW GUI. Simulation parameters are input by the user via the Simulation Parameters property sheet accessed from the Simulation->Set Simulation Parameters menu command. A simulation is run when the user selects Simulation->Run from the menu which causes the *IDM_SIMULATION_RUN* case of the *WndProc_OnCommand()* message handler function to be executed which calls the function *ContamSim()* located in the source file *SimDlgX2.c*. *ContamSim()* uses the *CreateProcess()* Windows API function to execute *contamx2.exe* in another thread. Complete documentation of the solver is provided in Part 2 – ContamX Program Documentation.

3.5.6 Viewing Simulation Results

The equation solver creates an output file having the same name as the project file but with the *.sim* extension. The GUI can be used to display these results. Results can also be exported to tab-delimited text files for importing into spreadsheet programs for more detailed analysis or written to report files that are organized for legibility. There are several methods of viewing results via the GUI as described below.

3.5.6.1 SketchPad Results

SketchPad results – in the form of color-coded lines to indicate the relative magnitude and direction of pressure drop and airflow rate – can be displayed for each airflow path one level and one time step at a time (See figure below). If a matching *.sim* file is available, a global variable *_resready* is set to true, and the user can select to view/hide the results display and select different time steps of transient results to display via the set of View menu commands which are handled by the *IDM_VIEW_X* cases of the function *WndProc_OnCommand()*. The SketchPad results are read from the *.sim* file by the *age_of_air()* function for the current time step as maintained in the global variable *_time_index*. The actual display of results is handled within the *SPWndProc_OnPaint()* message handler function which calls *res_dsp_level()* that performs scaling of the lines and calls lower level display functions *grlinevw()* and *grlinehw()* that in turn call Windows GDI functions to perform the actual graphical display. Along with the color-coded lines, the value and direction of the airflows and pressure drop of the individual airflow paths are displayed in the status bar for the currently highlighted airflow path icon on the SketchPad. This is done via the *sketch_status()* function which is called when several events occur including change of highlighted SketchPad cell via keyboard or mouse movement, changing the current level displayed or changing between SketchPad display modes (normal, results, wind and links).

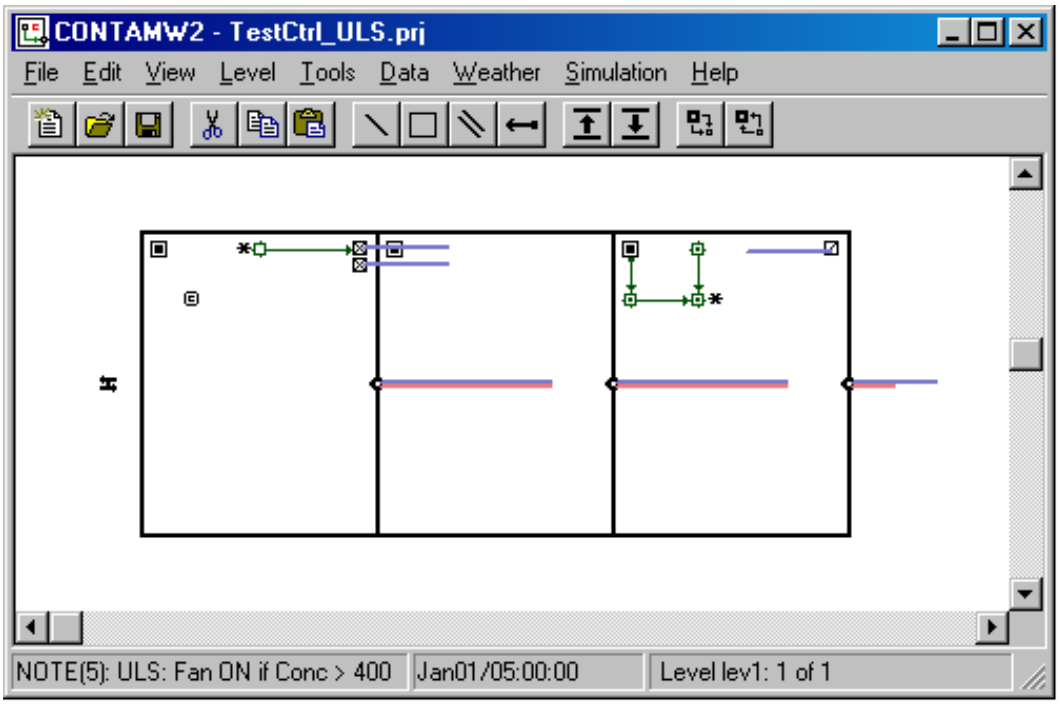


Figure – SketchPad Displaying Simulation Results of Airflow (blue lines) and Pressure Difference (red lines) for Airflow Paths

Function	File Name
WndProc_OnCommand( ) IDM_VIEW_X	WndProc.c
SPWndProc_OnPaint( )	SPWndPro.c
age_of_air()	flows_out.c
res_dsp_level( )	resultsw.c
grlinevw( ) grlinehw( )	wutils.c
sketch_status( )	bsketch.c

Table –SketchPad Results Display Functions

3.5.6.2 Results Display Window

Users may choose to display/hide a separate results display window (shown in the right side of the figure below). This window displays contaminant concentrations results and net airflow rates between adjacent zones of the currently highlighted zone icon on the SketchPad for steady state simulations and for the current display time step for transient simulations. Occupant related results can also be displayed in this window for highlighted exposure icons. This window is a modeless dialog box for which a global handle *g_hDlgResults* is maintained to which messages are dispatched from the main message loop. The dialog box procedure for this window *SSResultsDlgProc()* is located in the file *SSResDlg.c*. This window is updated when appropriate by *sketch_status()* which sends messages to this window via the *g_hDlgResults* handle.

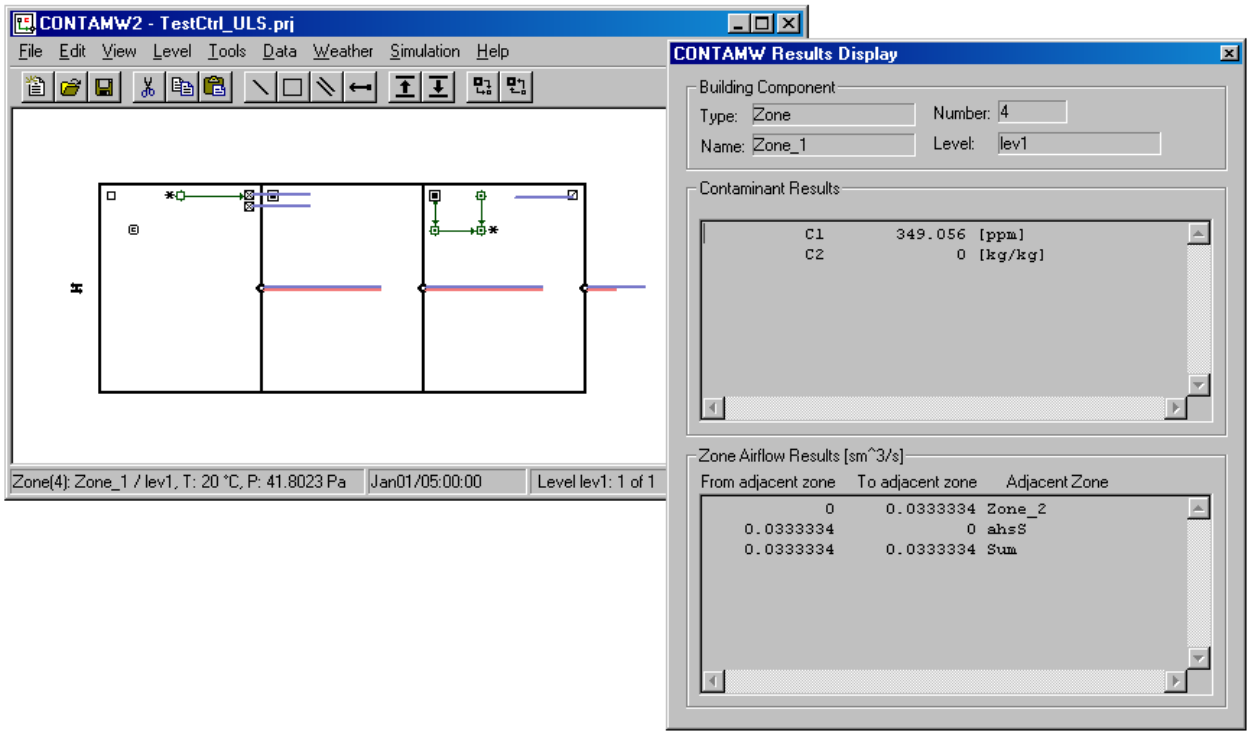


Figure –Results Display Window

3.5.6.3 Result Graphs

Users can use the GUI to display graphs of transient simulation results (see figure below) via the Simulation->Plot… menu commands. These commands are handled by the *IDM_SIMULATION_GRAPH_X* cases of the *WndProc_OnCommand()*, where *X* is either *CONTAM*, *AIRFLOW* or *EXPOSURE*. Each of these cases will in turn creates a dialog box having associated dialog box procedures *GraphRsltContamDlgProc()*, *GraphRsltAirflowDlgProc()* and *GraphRsltExposDlgProc()* located in the files *GrphCtm.c*, *GrphFlow.c* and *GrphExp.c* respectively. These procedures and others within these files provide for the user input of chart options, allocated memory for data to be plotted and creation and display of the windows in which the charts are displayed. The charts are created and displayed using third party charting software know as Olectra Chart. Olectra Chart is implemented as a DLL, *olch2d32.dll*, whose import library *olch2d32.lib* is included in the project along with other related header files: *oc_color.h*, *olch2d.h*, *olch2dcm.h*.

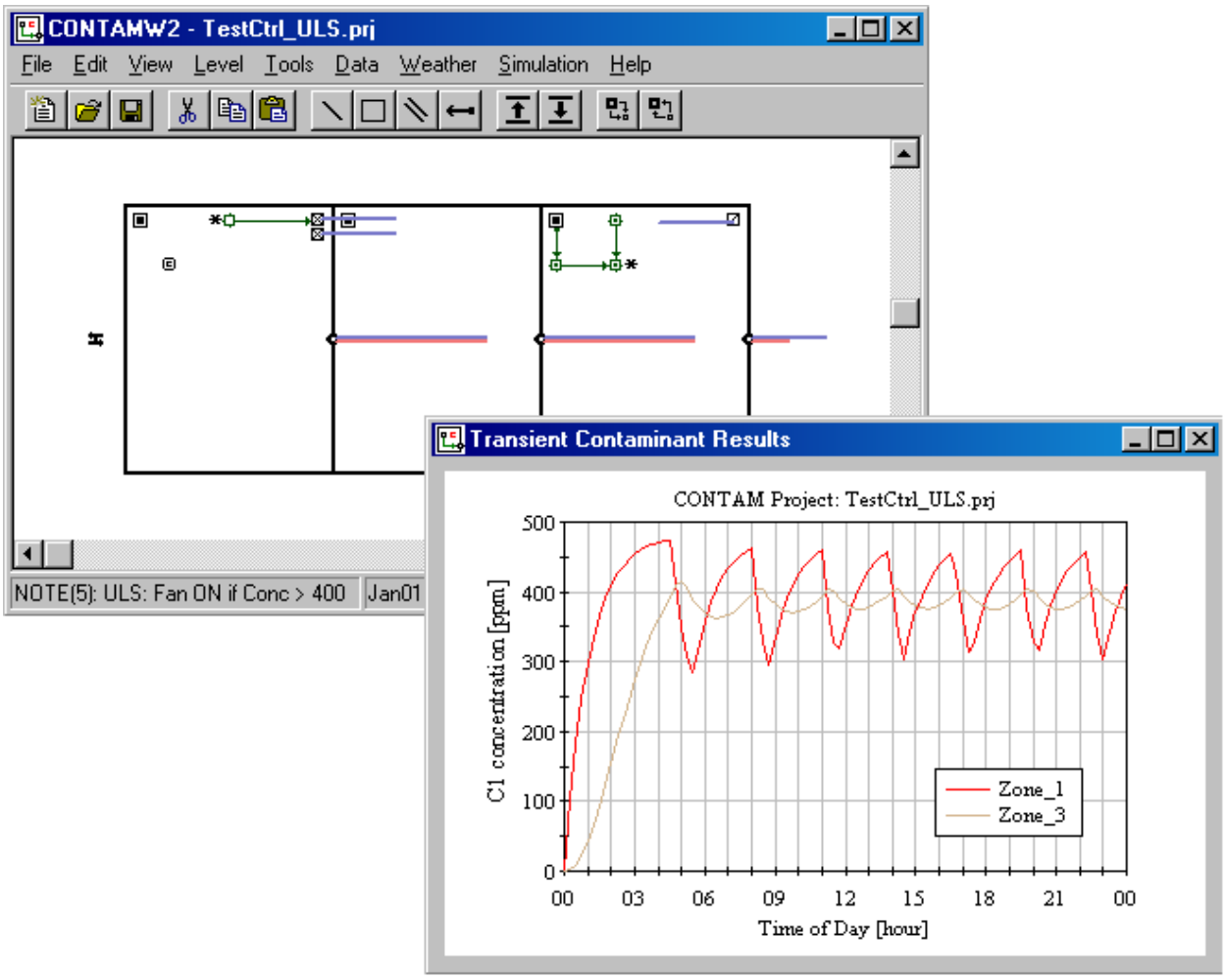
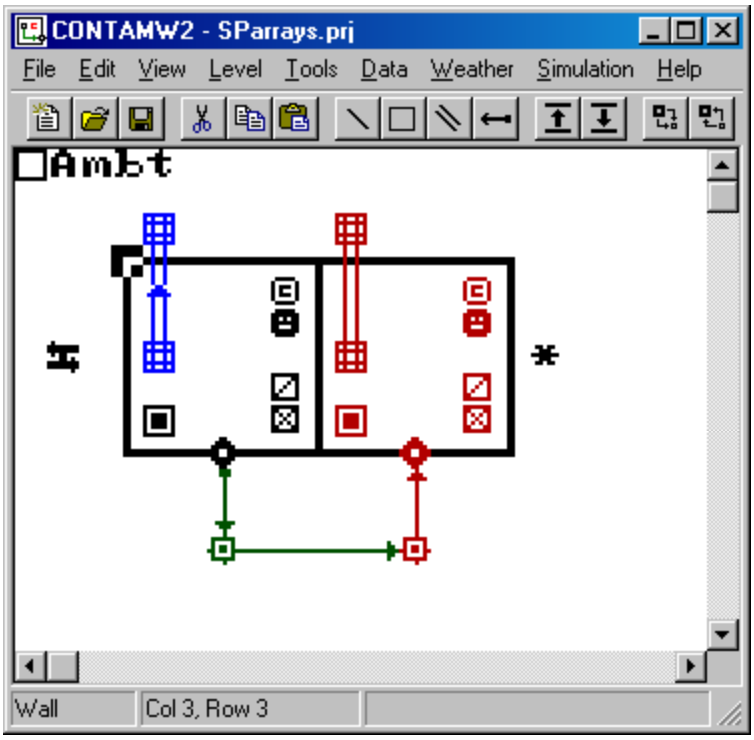


Figure – Transient Contaminant Results Graph

APPENDIX 3A

This appendix displays the SketchPad arrays presented in section 3.4.1. All of the spreadsheets presented in this appendix refer to the figure below, and each spreadsheet represents part of the arrays in which SketchPad data is stored. The associated project file, *SParrays.prj*, is installed in the samples subdirectory of the CONTAM 2.1 installation directory.



SketchPad used to demonstrate SketchPad array data.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	14	11	11	11	11	11	19	11	11	11	11	11	15	0	0
4	0	0	12	0	0	0	0	0	12	0	0	0	0	0	12	0	0
5	0	0	12	0	0	0	0	0	12	0	0	0	0	0	12	0	0
6	0	0	12	0	0	0	0	0	12	0	0	0	0	0	12	0	0
7	0	0	12	0	0	0	0	0	12	0	0	0	0	0	12	0	0
8	0	0	12	0	0	0	0	0	12	0	0	0	0	0	12	0	0
9	0	0	17	11	11	11	11	11	21	11	11	11	11	11	16	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

_Walls[][] array of SParrays.prj

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	133	0	0	0	0	0	133	0	0	0
5	0	0	0	0	0	0	0	131	0	0	0	0	0	131	0	0	0
6	130	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0
7	0	0	0	0	0	0	0	129	0	0	0	0	0	129	0	0	0
8	0	0	0	5	0	0	0	128	0	5	0	0	0	128	0	0	0
9	0	0	0	0	0	23	0	0	0	0	0	23	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

_Sketch[][] array of SParrays.prj

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
2	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
3	a	a	W	W	W	W	W	W	W	W	W	W	W	W	W	a	a
4	a	a	W	1	1	1	1	1	W	-2	-2	-2	-2	-2	W	a	a
5	a	a	W	1	1	1	1	1	W	-2	-2	-2	-2	-2	W	a	a
6	a	a	W	1	1	1	1	1	W	-2	-2	-2	-2	-2	W	a	a
7	a	a	W	1	1	1	1	1	W	-2	-2	-2	-2	-2	W	a	a
8	a	a	W	1	1	1	1	1	W	-2	-2	-2	-2	-2	W	a	a
9	a	a	W	W	W	W	W	W	W	W	W	W	W	W	W	a	a
10	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
11	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
12	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
13	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
14	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
15	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a

a = AMBT = 32766
W = WALL = 32767
ZNDF = -2 (undefined zone)

_Zones[][] array of SParrays.prj

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0
6	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	2	0	-2	0	0	0	0	0	0	0
9	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

_Paths[][] array of SParrays.prj

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	162	0	0	0	0	0	162	0	0	0	0	0	0	0
3	0	0	0	146	0	0	0	0	0	144	0	0	0	0	0	0	0
4	0	0	0	154	0	0	0	0	0	144	0	0	0	0	0	0	0
5	0	0	0	144	0	0	0	0	0	144	0	0	0	0	0	0	0
6	0	0	0	162	0	0	0	0	0	162	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

_Ducts[][] array of SParrays.prj

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

_Ctrls[][] array of SParrays.prj

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	181	0	0	0	0	0	177	0	0	0	0	0
11	0	0	0	0	0	179	0	0	0	0	0	169	0	0	0	0	0
12	0	0	0	0	0	185	170	168	168	168	168	174	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

_Links[][] array of SParrays.prj

REFERENCES

1. Emmerich, S.J., W.S. Dols, and J.W. Axley. *Natural Ventilation Review and Plan for Design and Analysis Tools*. NISTIR 6781, National Institute of Standards and Technology. 2001.
2. Dols, W.S. and G.N. Walton. *CONTAMW 2.0 User Manual*. National Institute of Standards and Technology. DRAFT 2002.
3. Axley, J.W. *Application of Natural Ventilation for U.S. Commercial Buildings*. GCR-01-820 NISTIR 6781, National Institute of Standards and Technology. 2001.
4. Axley, J.W. *Residential Passive Ventilation Systems: Evaluation and Design*. AIVC Technical Note 54. Coventry, AIVC.
5. ASHRAE 2001. *ASHRAE Handbook - 2001 Fundamentals*, Atlanta, GA.
6. Irving, S. and E. Uys. *CIBSE Applications Manual: Natural Ventilation in Non-domestic Buildings*. 1997, CIBSE: London.
7. Marion, W. and K. Urban. *User's Manual for TMY2s*. National Renewable Energy Laboratory, 1995.
8. Press, W.H., S.A. Teukolsky, W.T. Vetterling & B.P. Flannery. 1992. *Numerical Recipes in C: the Art of Scientific Computing, Second Edition*, Cambridge University Press.

ACKNOWLEDGMENTS

This effort was supported by the Naval Surface Warfare Center Dahlgren Division under Military Interdepartmental Purchase Request # N00178-02-MP-00374.